REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 10/7/97	3. REPORT TYPE AND DATES COVERED Final, 7/8/96 - 10/7/97

4. TITLE AND SUBTITLE A Highly Functional Decision Paradigm Based on Nonlinear Adaptive Genetic Algorithm	5. FUNDING NUMBERS DAAH04-96-C-0063
6. AUTHOR(S) Andrew Kostrzewski, Jeongdal Kim	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Physical Optics Corporation Engineering & Products Division 20600 Gramercy Place, Suite 103 Torrance, California 90501	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211	10. SPONSORING / MONITORING AGENCY REPORT NUMBER ARO 36350.1-EL-SB2

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In Phase II, the genetic algorithm (GA) developed in Phase I was refined and integrated with a neural network for network topology optimization. In addition, a Mathlink GA was developed as a Mathematica module. POC wrote the Mathematica plug-in module as a function optimizer using Mathlink. The GA route optimizer was written, tested, and demonstrated. On the hardware side, a TMS320C80-based parallel DSP board was used as a testbed for parallel GA. The GA programs running on the parallel computing hardware exhibited a significant speedup.

14. SUBJECT TERMS Network Topology Optimization, Mathlink, Mathematica Plug-In, GA Route Optimizer, DSP	15. NUMBER OF PAGES 70
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Significance

Progress in information technology over the past two decades has dramatically increased the amount of information that any given user needs to handle. Information workers are now presented with more information that they can hope to assimilate. The information technology explosion has confronted data users with incredible volumes of data, which mask rather expose the useful information that is required to make timely, intelligent decisions. This obviously further complicates the control of processing units that must be located remote for either security or cost reasons. This adds further complexity to large-military-system data management problems for all types of Army, Air Force, Navy, and Marine Corps needs. For example, Desert Storm required transportation of troops/equipment over long distances with large numbers of degrees of freedom such as departure/arrival schedules for ships, connecting flights, troops, and military equipment. Optimizing a transportation route in such a case requires considering a large number of connecting points. In fact, even in such a seemingly simple case the number of statistical problems is large.

Current GA techniques, though much faster than even the fastest non-GA convergence techniques, are not as fast as POC's *Fast Evolving Parallel Genetic Algorithm* (FEPGA) by an order of magnitude. This advantage becomes critical for large systems with more than 10 degrees of freedom. These degrees of freedom, which define the dimensionality of specific military systems, represent the space-time domain (x,y,z,t) plus a number of constraints specific to the given data management problem.

1.2 POC's Approach as Proposed

For fast decision making in an exponentially growing search domain, Physical Optics Corporation (POC) proposed for Phase I a universal decision module based on a FEPGA, which not only has an edge in convergence speed, but also has several additional features:

1. The FEPGA can learn the **historical convergence** (or generational evolution) between parents, between offspring, and between parents and offspring. This characteristic allows the FEPGA to search for a **global** and therefore robust solution, rather than local solutions. The FEPGA performs intelligent adaptive offspring selection, making it well suited for decision making in **rapidly changing** environments.
2. Through this adaptive selection process, the FEPGA eliminates the huge proportion of redundant offspring by means of a very simple computational fitness function. As a result, the FEPGA can afford to search a large data space or handle decision making for a large-dimensionality problem. In Phase II, we implemented a FEPGA that can handle problems of up to 20 dimensions through parallel processing.
3. The FEPGA can adjust the **rate** of evolution. *This is useful for time-dependent problems.* Responding to sensor inputs at variable rates, the FEPGA controls its sensitivity and adaptability. For example, in a rapidly changing environment the

FEPGA will speed up evolution to adapt quickly; when there is little change in the environment, the FEPGA can slow down evolution and concentrate on spanning the relationship of parents and offspring in the current generation. This allows the decision making system to increase its sensitivity to prepare for unexpected subtle changes in the environment.

4. The tunable rate of generation evolution is the key feature in the FEPGA. At initialization, crossover is the dominant factor, but its importance later decreases, maintaining the average convergence gradient (ACG) and conserving convergence time. The crossover and mutation rates are optimized using real-time ACG values. More specifically, if we observe degrading ACG we *reduce* the crossover rate. The rate is determined by fuzzy logic rules, which in turn are determined by the history of the convergence, the similarity of the parents, and the magnitude of the convergence gradient.
5. Conventional GAs have a fixed fitness function (i.e., a fixed boundary condition), which prevents them from adapting easily to rapidly changing environments. The FEPGA, on the other hand, has a tunable fitness function (i.e., an adjustable boundary condition), making it agile in a dynamic environment.

For Phase II, POC proposed to expand the FEPGA, integrating it into a parallel neural computing environment, and to investigate commercial application possibilities such as route optimization. In Phase II, POC did investigate these areas, with the results documented in this report.

The structure of this report is as follows. Section 2 describes the Phase II results . After a listing of the highlights of Phase II research, each category of the results is presented in detail in the order of description of software refinement, genetic neural network design, parallel GA with DSP, Mathlink version of GA optimizer, and other software development efforts. Section 3 describes software created in Phase II. The POC neural network generator is presented as well as a genetic neural network, Mathlink optimizer, and the new version of the function optimizer. Executable files for these programs are on the enclosed diskette.

2.0 PHASE II RESULTS

2.1 Highlights of Phase II Results

In Phase I, POC had demonstrated a universal decision making method based on GA and fuzzy logic. Specifically, POC completed the following:

1. Designed parallel genetic algorithm evolvers
2. Performed computer simulation
3. Determined the transfer function and population size
4. Maximized efficiency of mutation
5. Determined the adaptability of fuzzy rules to GA.

Based on these results, POC set several objectives for Phase II, including refining and optimizing the GA algorithm, designing a neural network using the fast-evolving fuzzy logic-based genetic algorithm, and designing and implementing the parallel multiprocessor computing platform. The three major milestones for Phase II research efforts set were:

1. Optimization of the nonlinear adaptive parallel computing paradigm;
2. Implementation of the high speed parallel multiprocessor computing platform; and
3. Integration of the parallel computing paradigm and the parallel computing platform into an adaptive neural network.

In order to meet these three milestones, POC established five Phase II technical objectives:

Objective 1. Optimize and refine the fast evolving fuzzy logic-based genetic algorithm. This will include applying the dynamic parallel programming method to genetic algorithms and completing the fuzzy rules that monitor and control each GA iteration.

Objective 2. Design a neural network using the fast evolving fuzzy logic-based genetic algorithm. This will include selecting a neural network model for a particular application based on selecting a problem domain and a network architecture. The main goal of this objective focuses on the adaptive training of a neural network using POC's fast evolving fuzzy logic-based genetic algorithm.

Objective 3. Design and implement the parallel multiprocessor computing platform. This will include designing a three-dimensional multi-digital signal processor (DSP) computing platform using commercially available DSP boards, interconnect topologies, and interfaces between the modules.

Objective 4. Implement an algotecture that is the integration of the developed algorithm and the platform. This will finalize the Phase II prototype implementation.

Objective 5. Optimize the Phase II prototype. This will include a demonstration of the Phase II prototype. An evaluation of the optimized prototype will be performed in order to develop a commercially viable high dimensionality decision making system for Phase III.

To meet Objective 1, the genetic algorithm written in Phase I was developed into a **general purpose scheduler** optimized with a fuzzified genetic algorithm. Although the scheduler problem looks simple enough for a human agent, its combinatorial complexity becomes daunting as the number of inputs increases. However, with its parallel and global search power, the GA performs combinatorial searches in problem spaces that are otherwise prohibitively large.

POC made crucial modifications to the design of the GA optimization package, so that it can be adapted readily to a broad range of application areas. POC also developed a "Mathlink" version of the optimization module to make it callable from Mathematica™, with the objective of making it more commercially attractive.

To meet the second objective, POC integrated a GA with neural networks (NNs). An example may clarify why this is useful. In a classification or financial prediction problem, where a neural network is trained with the data, a large number of variables affect the output, and the number of possible combinations among them is enormous. If a developer or user must try all the possible combinations of input variables, train a network, and check the results, it requires several days or even weeks just to identify the set of input variables that affect the result. Needless to say, as the number of inputs grows the number of possible combinations grows exponentially. If instead we use a genetic algorithm to find the best combination of input variables for NN training, we can avoid training countless neural network configurations that would be abandoned eventually anyway. Even though the problem sets for which genetic algorithms and neural networks are best suited are not the same, a genetic algorithm can facilitate the selection of the data set and NN architecture. For that reason, POC developed the genetic neural network described in more detail in later sections.

To meet Objective 3, POC implemented the GA in a parallel computing environment. We selected the TI 320C80 DSP, a fully programmable parallel processor, as the parallel computing platform. The processors on the C80 are connected by a crossbar network to on-chip SRAM and to a high speed external memory transfer controller for fast data transfer. This makes the use of shared memory efficient.

To meet Objectives 4 and 5, POC implemented a genetic algorithm in the parallel DSP environment, achieving a considerable speedup.

The highlights of Phase II are, in summary:

1. The GA program developed in Phase I was refined. POC repackaged the GA optimization modules as Dynamic Link Libraries that is callable from any Windows application.
2. The GA has been integrated with a neural network for network topology optimization. In this software development effort, POC used GA to evolve neural network structure to select the best combination and structure of neural network parameters.
3. A parallel DSP board was selected and installed, and a GA was tested on it. POC selected a TMS320C80 DSP board with a parallel computation architecture, and ran the GA program on it.
4. A Mathlink GA was developed as an external module of Mathematica. POC wrote the Mathematica plug-in module as a function optimizer using Mathlink.
5. The GA Route Optimizer was written, tested, and demonstrated. The GA route optimizer was implemented, and its usefulness for solving troop transportation problems was demonstrated.

These topics are discussed in some detail in the following sections.

2.2 Refinement of the Software

The initial GA program had many non-trivial limitations: The fact that it was DOS-based made it difficult to use. The list of fitness functions was predetermined, restricting its practical applicability. The dynamic range of each gene was limited to positive integer values. We revamped the whole software package to make it useful for virtually any application that needs an optimization module.

2.2.1 Introduction of Windows DLL Module

Windows™ DLLs are very similar in concept to DOS libraries. Unlike static DOS libraries, however, DLLs (Dynamic Link Libraries) are linked to the main program at run-time so that the routines can be used by one or more programs. During compilation and linking, a program finds the DLLs that contains the routines it needs. After the program is loaded, these routines are dynamically linked. The advantages of DLLs over static library routines is that they can be linked simultaneously to multiple applications.

In Phase II, POC repackaged the GA optimization modules in DLL format, separating them from the user interface module, and developed a GUI for the program. Another improvement in Phase II is the added capability for the user to select a preset fitness function or type in any other fitness function. Thus, the user can put frequently used functions into a data file, but still can use the system with any other custom-made function.

2.2.2 Description of GA Optimizer

In entering a function, the arguments must be called x1, x2, ..., up to a maximum of x10. Algebraic notation is used to enter an expression involving the arguments. Parentheses can be used, as can exponentiation. The built-in functions, such as *abs*, *sin*, *if-else*, are listed in Table 2-1. If a string is entered incorrectly, the program warns of a parsing failure.

Table 2-1. Built-in functions in GA Optimizer with Number of Arguments

abs	one	ifelse	three *
atan	one	ln	one
atan^2	two	log	one
ceiling	one	pi	no
cos	one	round	one
exp	one	sin	one
floor	one	sqrt	one

* x,y,z. Returns x,y,z

Once a function is selected or entered, the arguments are assumed to be bounded by 0 and 10, but the user can modify the range as needed. The search for extrema is carried out to a tolerance of ±0.1 on each argument, and the user can change the tolerance range as well. The bounds and tolerances are set by a dialog invoked by clicking the gene ranges button; the current gene ranges (argument ranges) are displayed in red in the upper right panel.

Optimization is initiated by clicking the Optimize button. For each iteration, the iteration count is displayed, as well as information regarding the actions taken by the genetic algorithm in that iteration.

While the optimization is proceeding, the Cancel button is available to abort a lengthy optimization. The final results of a converged optimization process are shown in the lower right panel. Multiple sessions can be initiated using File/New, though only one problem can be optimized at a time.

The File/Maximum Number of Iterations function, as its name implies, sets a maximum number of iterations. The GA Optimizer has been made easier to use, even for a user with minimal background in computing.

2.3 Genetic Neural Network Design

Genetic algorithms and neural networks are both modeled after biological systems in nature; the GAs imitate genetic evolution, and NNs mimic the brain. Each is suited to different types of problems. GA is primarily a search mechanism, testing out thousands of possible solutions to a problem and evaluating the results. For a GA to be applicable, some sort of model or function must be available against which to evaluate the output for each set of inputs the GA tries. Given an appropriate function for evaluation, the GA can find the best mix, best order, or best grouping.

A neural network tries to make sense of inputs and outputs by building some kind of internal model or function to connect them. This makes it good at pattern recognition and prediction based on data.

Given these properties, GAs can enhance neural networks. In Phase II, POC used GAs to find the best input combination for neural network training and the optimal neural network architecture for any given problem.

2.3.1 Evolving Neural Network

The powerful search capabilities of GAs can be combined with the learning capabilities of a neural network, using the GA to search through data to find the set of variables that will support the most accurate model, saving a great deal of training time. POC wrote software to automate much of the neural network design and development that a developer otherwise does by hand, by trial and error. These tedious tasks include testing/training data set selection, and determining which input variables to use. In our system, the GA is used to evolve neural network structures and select which input variables are significant. This evolving, learning, adapting artificial life capability is a powerful problem solving paradigm that can be used to meet many real world challenges.

This system was developed to meet the need to easily and quickly discover the best data elements and neural network architectures to build effective neural network applications. Many hours of human effort are spent attempting to find the best networks manually. It is clear that an effective automation tool is needed to off-load these hours of effort onto computers; POC applied GA techniques to this purpose.

2.3.2 Structure of a Genetic Neural Network

The program combines a GA and an NN so that the former determines the optimal structure of the latter. Specifically, it does the following:

1. Builds and validates training and test data sets
2. Creates a population of candidate input variables and neural structures
3. Builds the neural networks
4. Trains them
5. Evaluates them
6. Selects the best networks, in terms of some fitness function
7. Pairs up the genetic material representing the inputs and neural structure of these networks, and exchanges genetic material between them
8. Puts in a few mutations for a flavor of random search
9. Goes back into the training/testing cycle again.

This continues for a defined number of generations, for a defined period of time, or until an accuracy goal is reached (see Figure 2-1).

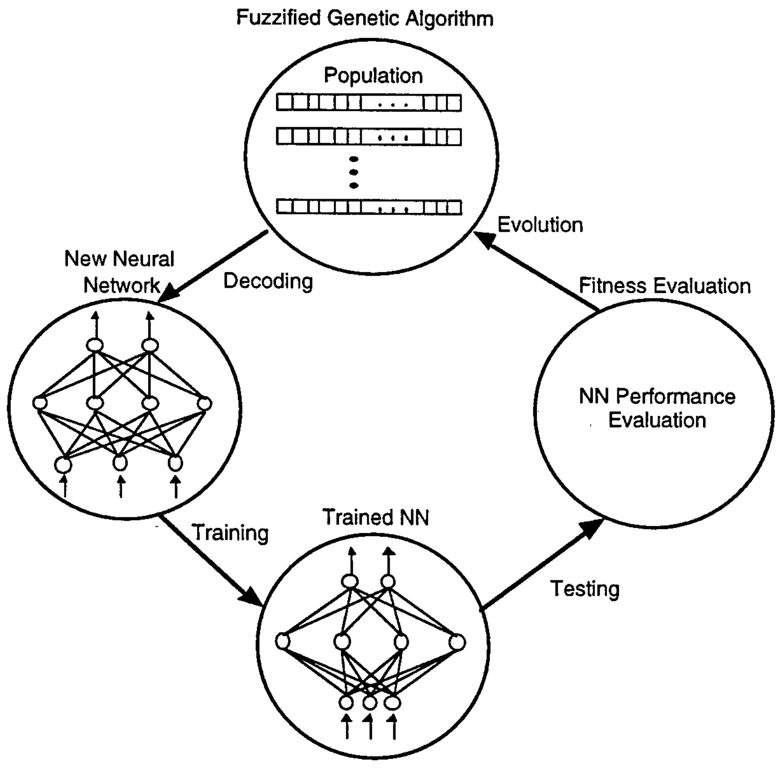


Figure 2-1
Genetic neural network system.

We represent the architecture of a network of N units by a connection control matrix C of dimension N × (N+1) (see Figure 2-2). The first N columns of matrix C represent the connectivity relationships among units in the neural network, and the final (N+1) columns store the threshold biases for the unit. For example, in Figure 2-2 a unit that receives two inputs will have a threshold bias of 1, and otherwise 0.

		Origin Node					
		1	**2**	**3**	**4**	**5**	**bias**
Destination Node	**1**	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	1	1	0	0	0	1
	4	1	1	0	0	0	1
	5	0	0	1	1	0	1

000000000000110001110001001101

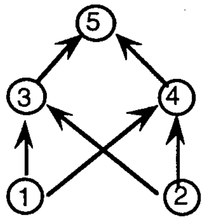


Figure 2-2
Conversion from connection control matrix at top, to bit strings, center, to network architecture at bottom.

Each entry $C_{i,j}$ in the matrix C is a member of the connection control set S, and indicates the nature of the connection from unit to unit. Thus, column i of C represents the fan-out of connections from unit i. Similarly, row j represents the fan-in of connections to unit j. The bit string in the middle of Figure 2-1, which has been created by the concatenation of successive rows of matrix C, is the population to be processed by the genetic operators.

Note that many different neural network architectures are implemented in a given generation. We automate neural network design by two adaptive processes: genetic evolution through generations of network architecture spaces, and back propagation learning in individual networks to evaluate the selected architectures. Thus, cycles of learning in an individual architecture are embedded within cycles of evolution in populations. Each learning cycle presents an individual neural network with the set of input and output pairs that define the task. The back propagation learning algorithm then adjusts the network connection weights so that it performs the input/output mapping task with increasing accuracy. Each evolution cycle evaluates one population of network designs according to their associated fitness values to yield an offspring population of more highly adapted network designs.

Back propagation is computationally intensive, but the effectiveness of genetic algorithms' combinatorial search capabilities would be difficult to overstate. For example, a problem consisting of finding the best combination (subset) of 20 inputs and up to 15 hidden nodes in a back propagation neural network has over 16 million permutations. To train each network in a hard, full search would be an appropriate project for a supercomputer, but with genetic algorithms a very good solution often appears in less than 1500 evaluations, which is less than one ten-thousandth of the total possible configurations. With the help of some statistical data analysis, highly fit networks are often found in the first generation evaluated. This is clearly an efficient means for discovering effective network structure/input combinations.

Note that by the nature of genetic algorithms these networks are not necessarily optimal, but do typically represent good solutions.

2.4 Parallel GA with DSP--Selection of a Commercially Existing DSP Board

POC used the Texas Instruments (TI) TMS320C80 processor integrated with a Matrox Genesis board from Matrox Imaging Products Group (see Figure 2-2), a complete PC/AT plug-in board for parallel processing. It supports four 32-bit parallel processors (PPs) and one 32-bit master processor (MP), all connected by a crossbar network. Additional C80 nodes, each with one MP and four PPs, can be mounted on the main board. This is one reason POC has selected the C80 for GA parallel computing. More processing power can be added easily depending on the processing power requirements for the particular application. It can be configured as SIMD (single instruction, multiple data) or MIMD (multiple instruction, multiple data). Therefore, a flexible system configuration can be designed taking into consideration cost, speed, and other parameters specific to each application.

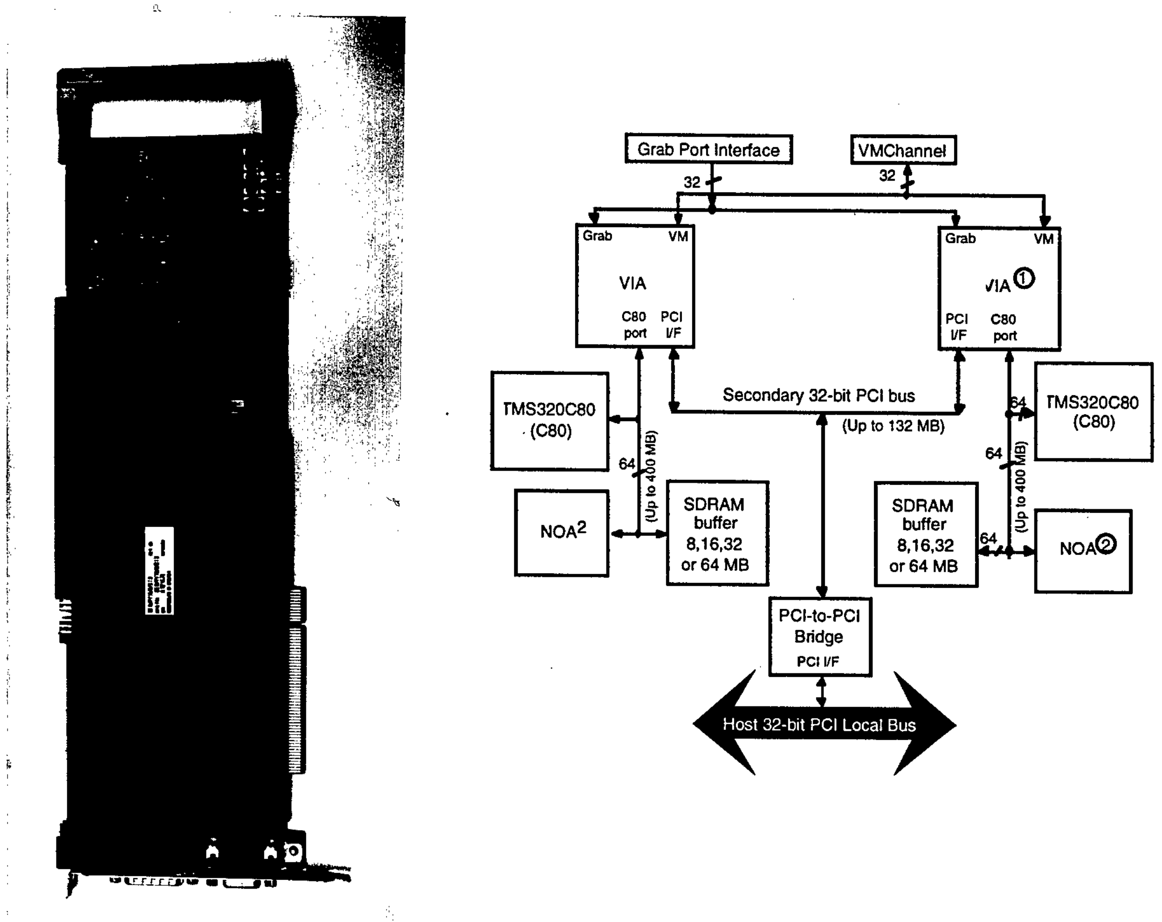


Figure 2-3
TMS320C80 processor board.

The major advantages of using the TMS320C80 DSP platform for parallel computing are:

- Not bus dependent
- Standardized building block for multiprocessing
- Wide variety of modules available for specific applications
- Upgradable performance at low incremental cost.

The four communication ports of the TMS320C80 offer a wide range of connection possibilities. It has a 32 bit address and data bus, and operates at 50 MHz, performing 100 MFLOPS.

Figure 2-3 shows the C80 processor board configuration. The high-performance link connecting acquisition, display, and processing is through the VIA (Video Interface ASIC), a powerful interface for the input port and the C80 connection, the SDRAM, and a PCI master/slave bus interface. The wide range of Matrox Genesis modules includes: general purpose modules carrying SRAM, multi-C80 modules; and application-specific modules, which carry a C80 and an I/O interface. All of the modules connect directly to the PCI bus interface of the Pentium PC motherboard.

Each C80, which can communicate with the main board through the VIA and in turn through the PCI-to-PCI bridge, has the following features:

- One 32-bit RISC master processor with integral FPU
- Four 32-bit integer advanced parallel processor DSPs
- 32 kbytes of internal RAM shared among processors (expandable to 50 kbytes)
- Crossbar for optimal internal connectivity
- Transfer controller for high performance external I/O
- 50 MHz system clock
- Internal FPU capable of 100 MFLOPS
- Up to 2 billion RISC-like operations per second
- 2.4 Gbytes/second sustainable on-chip data transfer rate
- 400 Mb/s off-chip peak transfer rate

Layer-to-layer interconnection is available between C80 processor boards.

2.5 Mathlink Version of GA Optimizer

POC's commercialization efforts for the GA Optimizer have been fruitful; a mathematical library that links the optimization module to Mathematica™ was written and favorably received by Wolfram Research, Inc. A sample run of the GA module in Mathematica follows, where bold face indicates what the user types in and normal Courier font shows that response from Mathematica:

```
link = Install ["e:\\pocsoft\\minimax"]
LinkObject [e:\pocsoft\minimax, 2, 2]

Minimize2 ["sin(x1_ -x2*cos(x2)", -10.0, 10.0, 0.1,
               -10.0, 10.0, 0.1]
{-1.5625, -9.53125, -10.4772]

?Minimize2
Minimize2 [f_String, L1_Real, T1_Real, L2_Real,
   U2_Real, T2_Real minimum value of the 2-argument function
   f with sets of the lower bounds, and the tolerances:
   Property of Physical Optics Corporation.

Uninstall [link]
e:\pocsoft\minimax
```

2.5.1 Description of Mathlink Version of GA Optimizer

To use the GA Optimizer in Mathematica, the user copies the executable file "minimax.exe" that accompanies this report to any designated directory, starts Mathematica, and then makes a connection to the file by entering

```
link = Install ["e:\\pocsoft\\minimax"]
```

This assumes that "minimax.exe" resides in the "e:\pocsoft" directory. Once the link is made, one can find the minimum value of a function with up to ten arguments. The first-order function is used here for the purpose of illustration:

```
MiniMize1 ["first_order_function here", L1, U1, T1]
where     L1:   the lower bound of the argument
          U1:   the upper bound of the argument
          T1:   the tolerance
```

Similarly, the user type the following command for a second-order function:

```
MiniMize2 ["second_order_function here", -10.0, 10.0, 0.1, -10.0,
10.0, 0.1]
```

The output is given in the form of a list such as:

```
{value_of_arg1_at_minimum,
value_of_arg2_at_minimum,
the minimum value}
```

One can find the maximum value of the function and the values of the arguments at that point in a similar way by calling the "maximize" function:

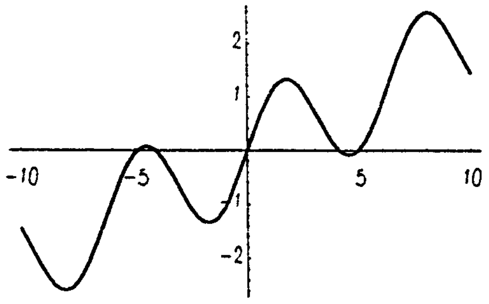
```
Maximize2  ["200 - (x1 ^ 2 + x1 - 11) ^2 - (x1 + x2 ^ 2 -7) ^2",
           -6.0, 6.0, 0.1, -6.0, 6.0, 0.1]
```

Multiple search sessions are possible. After completion of the task, the connection is broken by entering:

```
Uninstall [link]
```

Figure 2-4 shows a plot of a function and its minimum value as found by a GA Optimizer routine called from within Mathematica.

```
In[6]:=
   Plot[Sin[x] + x/5, {x, -10, 10}]
```



```
Out[6]=
   -Graphics-
In[7]:=

   MiniMize1["sin(x1) + x1/5", -10.0, 10.0, 0.01]
Out[7]=
   {-8.05664, -2.59086}
```

Figure 2-4
Mathlink version of GA Optimizer at work.

2.5.2 Advantages of POC Optimizer Module

One of the major features of a genetic algorithm is its ability to find a globally near optimal value without falling into local optima. While Mathematica™ can search for the global maxima and minima only for linear functions, the GA Optimizer does so for virtually any type of function.

Furthermore, POC's Mathlink version of the GA Optimizer includes 10 routines for finding maximum values: MaxiMize1, ..., MaxiMize10. Mathematica has no corresponding functions.

2.6 Other Software Development

2.6.1 Troop Transportation Global Optimization Problem

Consider a large-volume, high dimensionality, military data management problem; Troop Transportation Global Optimization, for which the time to reach an optimal solution increases logarithmically with the size of the problem (or with the number of degrees of freedom). Using FEPGA, however, the base of the logarithm is reduced step-by-step, so that convergence time depends much less critically on the size of the problem. This is because the rates of crossover, mutation, and reproduction are adaptable, controlled by the chromosome pool first order differential, which is a new internal parameter of the GA system. POC's GA system is parametric, i.e., the crossover, mutation, and reproduction rates are controlled internally by the convergence process.

2.6.1.1 Description of the Problem

The Troops Air Transportation (TAT) double-sorting global optimization problem belongs to the general class of time-scheduling problems. Consider the example TAT problem shown in Figure 2-5. At the start, the routing paths can be chosen almost arbitrarily, with, say, fixed connection points (such as A, 2, 7, 10, 15, Q). Each flight path is organized as a binary stream, or gene. For K planes, a single statistical realization is represented by K flight paths, so a chromosome is K-dimensional.

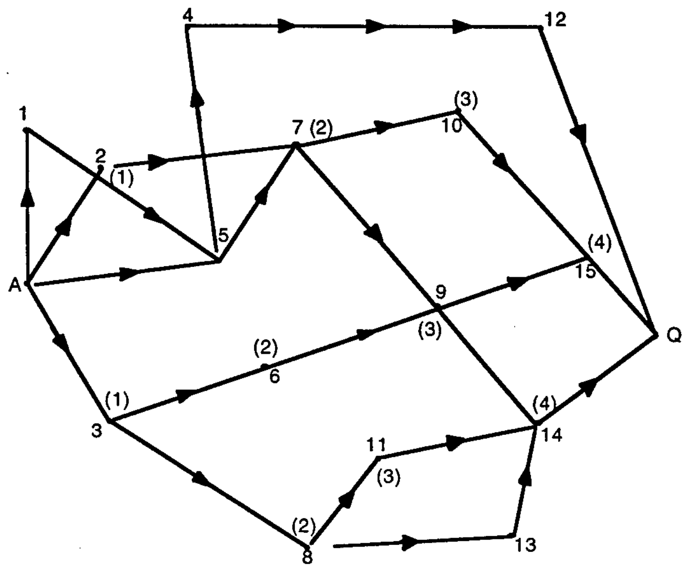


Figure 2-5
TAT through 17 airports, from Port A to Port Q.

Each port (except A and Q) is numbered by a digit 1 through 15. The numbers in brackets determine the specific flight, such as: A, (1), (2), (3), (4), Q, where four connecting ports are always assumed. Figure 2-5 is only a spatial coordinate map; i.e., the schedule is not shown.

2.6.1.2 Analysis of the Problem

In order to demonstrate the feasibility of using FEPGA for global optimization of troop transportation, we first perform problem quantization, or chromosome pool generation, and define the fitness function. We make a number of simplifying assumptions, not to make the problem simple to solve (since much more complicated problems can be solved using the FEPGA), but rather for clarity of explanation.

We make the following simplifying assumptions:

- We transport only troops (i.e., single soldiers).
- We transport only through the air (using airplanes).
- Although the military units are transported through a variety of airports (or ports), all units are transported from port A to port Q as shown in Figure 2-5. For simplicity, a constant number of four connecting ports is always assumed.
- The military units transported are sufficiently small (e.g., platoons) that they are not partitioned for flight.

In spite of these simplifying assumptions, considerable flexibility is still implemented in the program, such as variable numbers of units and troops, variable numbers of ports, planes, and their locations, and variable plane speeds. Also, very flexible time schedules based on the "window" concept are assumed.

The chromosome space (pool) is separately constructed from the travel path for each plane as follows:

1) q-segmented numbering is introduced; e.g., the example travel path shown in Figure 2-5 is translated into the following q-sequence:

$$\begin{tikzcd}
(A, \arrow[d, leftrightarrow] & 2, \arrow[d, leftrightarrow] & 7, \arrow[d, leftrightarrow] & 10, \arrow[d, leftrightarrow] & \text{Regular notation)} \arrow[d, leftrightarrow] \\
(A, & 1, & 2, & 3, & \text{q-sequence)}
\end{tikzcd} \qquad (2\text{-}1)$$

2) All travel paths' regular notations are organized in sequence according to arithmetic value; i.e., for two paths, as shown in Figure 2-5:

(2, 7, 10, 15) and (3, 6, 9, 15) (2-2)

We construct two integers:

271015 and 36915 (2-3)

with larger integers following smaller.

3) A unique sequential number is attached to each integer. This route is shown on the basis of a simple example. Assume only four connecting ports, three of which are

used for each travel path. If these ports are numbered 1, 2, 3, and 4, then the following travel paths are possible:

$$\begin{array}{cccc} 123 & 124 & 234 & 134 \\ 132 & 142 & 243 & 143 \\ 213 & 214 & 324 & 341 \\ 231 & 241 & 342 & 314 \\ 312 & 412 & 412 & 413 \\ 321 & 421 & 421 & 431 \end{array} \qquad (2\text{-}4)$$

They are organized in sequence:

$$\begin{array}{ccccccc} 123 & 124 & 132 & 134 & 142 & 143 & 213 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ (1) & (2) & (3) & (4) & (5) & (6) & (7) \end{array} \qquad (2\text{-}5)$$

This is a 24-number sequence.

4) The organized sequence is presented in the form of a binary stream; e.g.,

$$\begin{aligned} 22 &= 0 + 2^5 \times 1 + 2^4 \times 0 + 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \\ &= 0 + 16 + 0 + 4 + 2 + 0 \\ &= (010110) \end{aligned} \qquad (2\text{-}6)$$

5) Each flight path is organized as a binary stream, or gene. Therefore, for a single realization represented by K flight paths (some of them identical), equivalent to K planes, we obtain K genes, so a chromosome is K-dimensional.

The maximum number of possible flight paths is:

$$\frac{(1/2)N!}{(N-P)!} \qquad (2\text{-}7)$$

where N and P are integers, n! = n(n-1) (n-2) ..., N is the total number of ports, and P is the number of connecting ports to be used for each flight. The (1/2) comes from the fact that all flights are one-way. Since every flight path can be realized in a number of ways, as in Eq. (2-7), and the number of possible paths is equal to the number of planes K, the total number of realizations (or number of degrees of freedom) is:

$$D = D(N,K;P) = \left[\frac{(1/2)N!}{(N-P)!}\right]^K . \qquad (2\text{-}8)$$

For example:

N=50, P=4, K=200; then Eq. (2-8) evaluates to 101288, which is far to large for conventional sorting.

2.6.1.3 Cost Function for the Troops Transportation Double-Sorting Global Optimization Problem

The Troops Transportation Cost Function (T2CF) is defined as follows:

$$\begin{aligned}
T^2CF = & Wt\sum_K \sum_q \frac{EDnmk}{V''} + Wd_1\sum_i \left(MAX,[0,TA'_i - tA_i]\right) \\
& + W_{d''}\sum_i \left(MAX[0,tA_i - TA_{i''}]\right) + WA'\sum_i \left(MAX[0,TQ_{i'} - t_{Q_i}]\right) \\
& + W_{A''}\sum_i \left(MAX[0,tQ_i - TQ_{i''}]\right) + W_{A'}\left(MAX\left[0,\sum L_{ik} - C_k\right]\right), \qquad (2\text{-}10) \\
& + W_R\left(MAX[0,R_{ik} - MRT]\right) + Wp\left(MAX\left[0,\sum L_{kn} - Cn\right]\right)
\end{aligned}$$

where the critical weighting factors are:

W_t	Travel weighting factor
$W_{d'}$, $W_{d''}$	Departure schedule penalty weighting factors
$W_{A'}$, $W_{A''}$	Arrival schedule penalty weighting factors
W_a	Penalty weighting factor for capacity of the plane
W_p	Penalty weighting factor for capacity of the port
W_R	Penalty weighting factor for maximum allowable travel time of troops.

These penalty factors should be adjusted according to the importance of a given constraint. For example, if a not "too-late" arrival time is more critical than a not "too-early" arrival time, then

$$WA'' > WA'. \qquad (2\text{-}11)$$

The following term is a quadratic bracket:

$$MAX[0, H], \qquad (2\text{-}12)$$

which is either 0 if a given constraint is satisfied, or H if this constraint is violated.

The chromosome space is now tested against the T2CF cost function value, and the small T2CF values are promoted by using the FEPGA route.

The next section outlines a preliminary version of the transportation scheduler that POC built.

2.6.2 GA Route Scheduler

To demonstrate the feasibility of using a genetic algorithm for a transportation optimization problem, POC developed a scheduler that finds the shortest route through a set of cities. This demonstration program made use of a Los Angeles area map, on which a number of cities were identified as candidate "ports." The initial state of the program is shown in Figure 2-6.

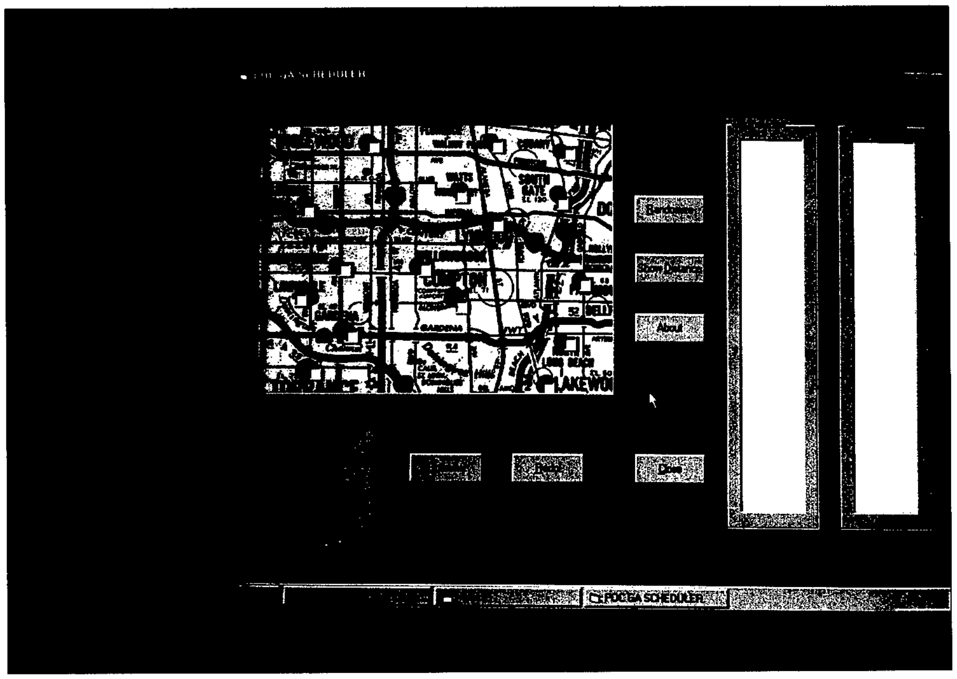


Figure 2-6
Initial state of route optimization problem

In this program, a map is displayed with a number of checkboxes for selection. Each checkbox is associated with one of 16 cities. The user can select the cities he/she needs to visit by checking the checkbox. The number of cities checked is displayed in the topmost textbox, the accumulated route length in the second box, the optimized route length after optimization in the third box, and the mileage saving in the last box. At the right of the screen, the original itinerary and the optimized one are displayed. Figure 2-7 shows eight cities selected.

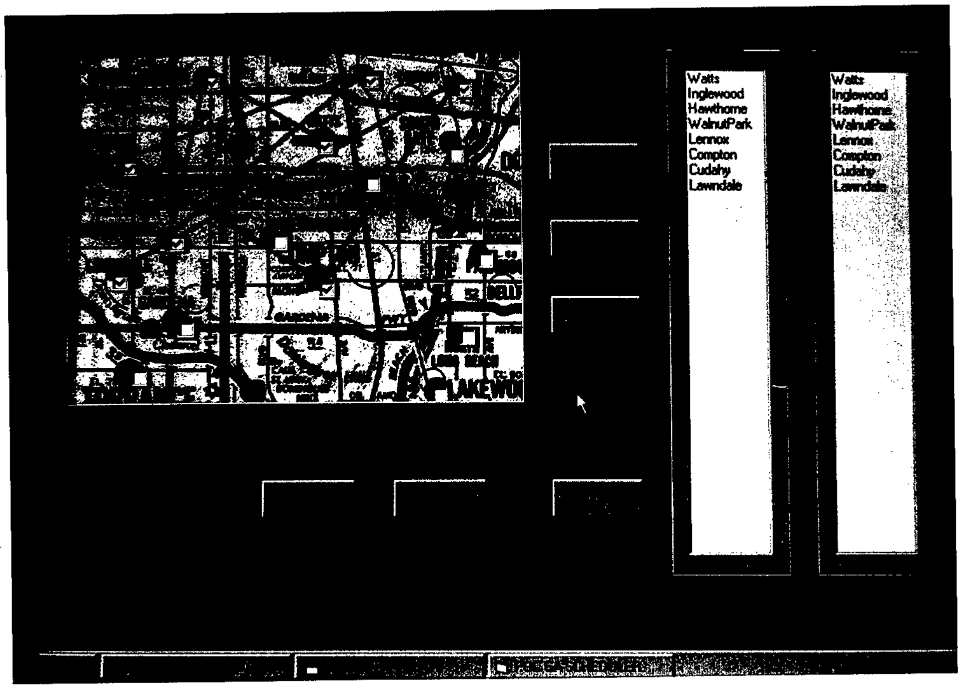


Figure 2-7
Input selection in the GA scheduler

In Figure 2-7, the textboxes reflect the changed statistics. The route length for the selected cities is shown, with their names. The user now can click on the Optimize button and start optimization.

The stopping condition can be defined in any of a number of ways. In this program, the number of iteration since the last improvement is compared with a preset threshold value. As long as the stopping condition is not satisfied, a new mating pool is created based on the performance of the previous population and crossover and mutation generate a new population, which then goes back to the cycle. When the stopping condition is met, the optimization process ends and the program displays the result as in Figure 2-8, which shows the optimized route and other statistics. At this point, the user can start another round of optimization by clicking Redo, or can exit the program by selecting Close.

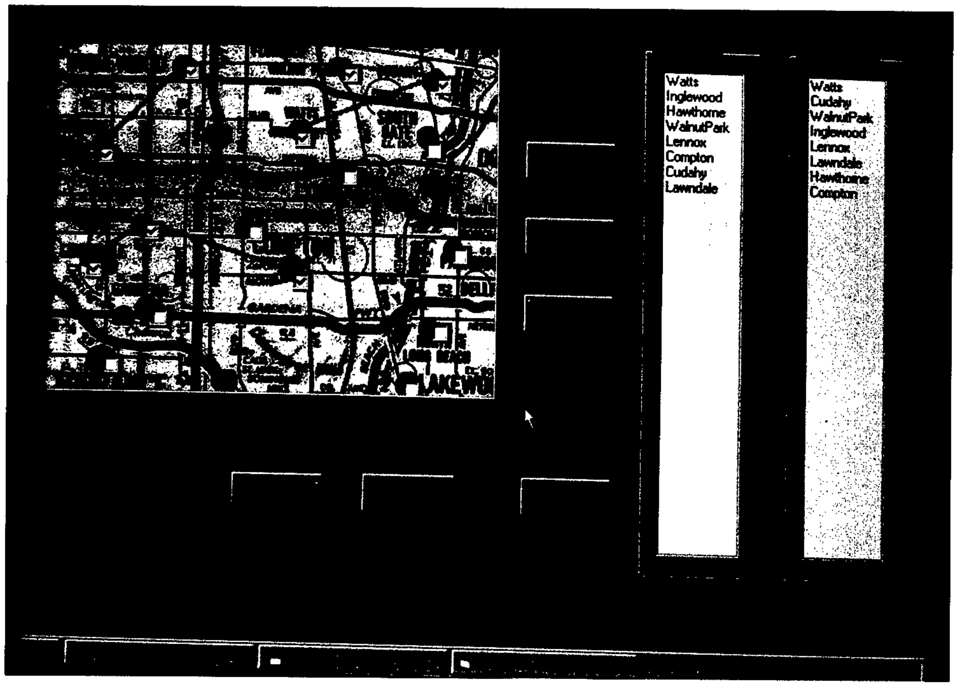


Figure 2-8
Final result of optimization.

3.0 PHASE II DEMONSTRATION

3.1 Network Generator

POC's Fuzzified Evolving Parallel Genetic Algorithm (FEPGA) can find the best neural network structure for a given problem. The FEPGA generates various neural network structures in accordance with a number of fitness function generators. The neural network is trained on the input data of the selected problem. The minimum squared error that guides the convergence of the neural network is used as the fitness measure. The best structures found are input to the FEPGA for evolution. The top level algorithm flow is as follows:

```
Create a population
Initialize the population
for i = 1 to max_iterations
    for j = 1 to population_size
        Decode chromosome
        Build neural network
        Train/evaluate neural network
        Assign some evaluation score to the network performance
        If it's a good one, then keep it in a list
    next j
    Selection
    Pair mates
    Crossover
    Mutate
next i
```

3.1.1 Description of the XOR Problem

As a simple case to demonstrate a working neural network system, POC used the much-studied exclusive OR (XOR) problem. This is a linearly non-separable problem that cannot be solved without using hidden units. The input and the expected output for XOR is shown in Table 3-1.

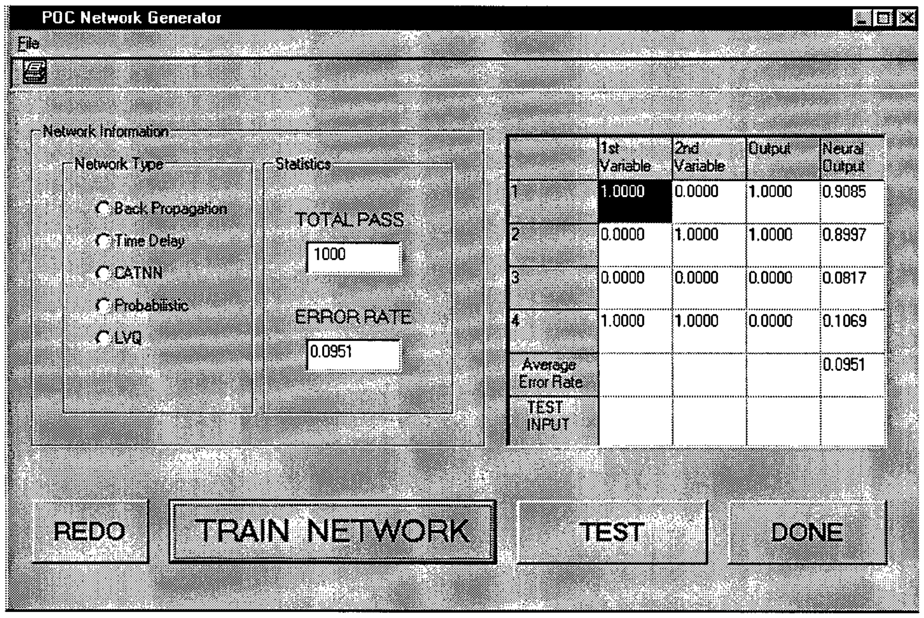
Table 3-1 Input and Output of XOR Problem

Input	Output
0 0	0
0 1	1
1 0	1
1 1	0

The XOR function maps two binary inputs to a single binary output as follows: 00->0, 01->1, 10->1, 11->0. The XOR problem has been tested on the POC network builder (POCNET).

3.1.2 POC Neural Network Generator

The expanded POC Neural Network Generator interface is shown in Figure 3-1.



TOTAL PASS: 1000

ERROR RATE: 0.0951

	1st Variable	2nd Variable	Output	Neural Output
1	1.0000	0.0000	1.0000	0.9085
2	0.0000	1.0000	1.0000	0.8997
3	0.0000	0.0000	0.0000	0.0817
4	1.0000	1.0000	0.0000	0.1069
Average Error Rate				0.0951
TEST INPUT				

Figure 3-1
Expanded Neural Network Generator.

3.2 Genetic Neural Network

3.2.1 Population Generation

The genetic algorithm generates a population with parameters that will be used in neural network development. The network generation module of the POCNET must deal in variables such as the number of inputs, number of hidden layers, number of hidden units, type of transfer function, and the number of outputs. The XOR problem, which we have taken as our test case, has two inputs and one output. The "number of inputs" is the number of variables that are fed to the neural network, and the "number of records" is the total number of input data records used for training. The number of hidden layers is usually one or two, depending on the complexity of the network. For the XOR problem, four or five hidden units should be enough. The transfer function is a tangent function or a logistic function, so a single bit is sufficient to represent it. The number of outputs varies from problem to problem, but the output is always a single one-bit unit.

In sum, one string member of the population can be represented as:

$$\underbrace{1010}_{\text{number on inputs}} \quad \underbrace{01}_{\text{number of hidden layers}} \quad \underbrace{001}_{\text{number of hidden units}} \quad \underbrace{01}_{\text{type of transfer funciton}} \quad \underbrace{01}_{\text{number of outputs}}$$

The total number of bits used to represent the gene here is 13. The population pool can contain few or many genes, depending on the complexity of the problem.

3.2.2 Mapping of Genotype to Phenotype

"Genotype" in artificial genetic systems refers to the total package of strings, or to individual strings. The genotype is decoded to form a particular parameter set, or alternative solution. The designer of an artificial genetic system has a variety of alternatives for coding both numeric and nonnumeric parameters. Our task here is to map the string (genotype) to a neural network structure (phenotype). To do so we use parameters such as neural network type, number of hidden layers, total number of nodes, and transfer function. Neural network design using genetic algorithm has been reported in a number of papers [1-7]. Not only does the current work differ from theirs in algorithmic details, but it also focuses on actual implementation of the paradigm.

The number of hidden layers can be 0, 1, or 2. The number of hidden neurons in a layer can vary from 1 to 256. The transfer function can be logistic sigmoid, hyperbolic tangent, or linear. A chromosome can encode four parameter as shown in Figure 3-2.

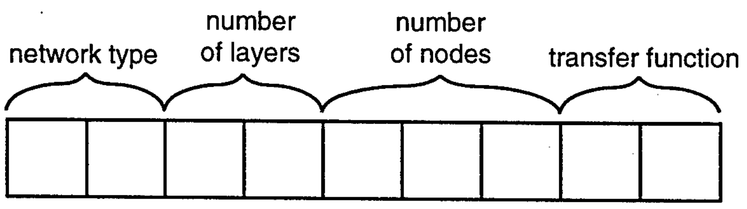


Figure 3-2
Structure of a chromosome.

For example, after initialization a GA population could have the following set of chromosomes:

011010101
101010101
110111010
111010111

This genotype could then be decoded to the corresponding phenotype:

011010101	⟶	BP, 2, 5	logistic sigmoid
101010110	⟶	BPNN, 2, 5	hyperbolic tangent
110111010	⟶	LVQ, 1, 6	hyperbolic tangent
111010111	⟶	LVQ, 2, 5	linear

3.2.3 Fitness Function

The performance of a given neural network determines the fitness value assigned by the genetic algorithm, represented by the average error rate. In this program, two types of transfer function are used: a tangent function and a logistic function. How the error is calculated depends on the neural network type used.

The hyperbolic tangent function is used as a transfer function to calculate the output y for each node as follows:

$$y = \frac{\left(1 - e^{-D}\right)}{\left(1 + e^{-D}\right)}, \qquad (3\text{-}1)$$

where

$$D = w_0 + \sum_{i=1} w_i x_i, \qquad (3\text{-}2)$$

where w_0 is the bias of the node, each w_i is a weight for the connection from the i-th node of the previous layer, and x_i is the input from the i-th node of the previous layer. For estimating purposes, it is standard practice to use only linear transfer functions in the nodes of the output layer.

The logistic function is expressed as follows:

$$y = \frac{1}{\left(1 - e^{-D}\right)}, \qquad (3\text{-}3)$$

where y is the output of the logistic function, the x_i's are the inputs, and the w_i's are the free parameters. D is defined as in Eq. (3-2).

The neural network performance P for each input is calculated by the selected method, and the error rate E is found by comparing P with the desired output O. Then the average error rate A is given by the following simple formula:

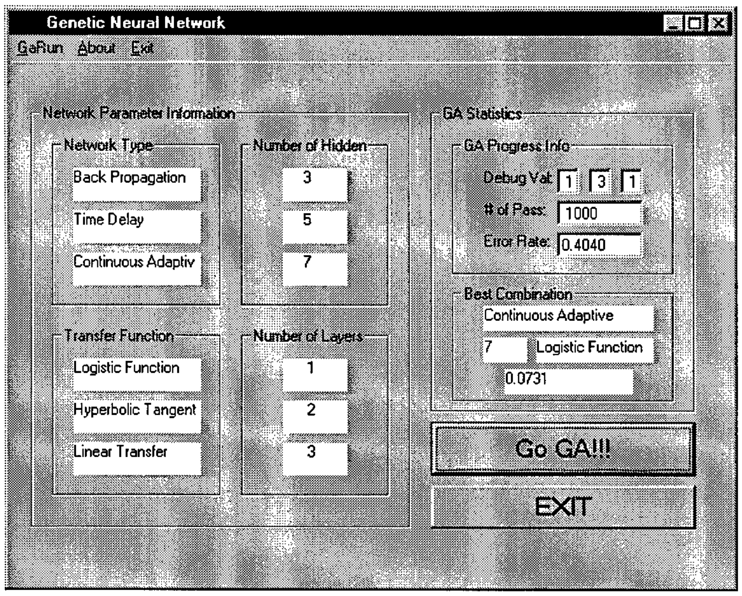
$$A = \frac{\sum_i E}{\text{number of records}}. \qquad (3\text{-}4)$$

3.2.4 Genetic Operation of GNN

Each gene -- i.e., each neural network architecture in our program -- that performs well remains in the population. After the neural network module is executed, pairs of genes are mated and undergo crossover and mutation.

3.2.5 Genetic Neural Network

The first version of the genetic neural network (GNN) interface is shown in Figure 3-3 and the corresponding executable is included on the attached diskette.



Network Parameter Information

Network Type: Back Propagation, Time Delay, Continuous Adaptiv

Number of Hidden: 3, 5, 7

Transfer Function: Logistic Function, Hyperbolic Tangent, Linear Transfer

Number of Layers: 1, 2, 3

GA Statistics

GA Progress Info

Debug Val: 1 3 1

# of Pass: 1000

Error Rate: 0.4040

Best Combination: Continuous Adaptive, 7, Logistic Function, 0.0731

Figure 3-3
POC genetic neural network.

The GNN interface consists of menus, three frames, and two control buttons, each of which is explained below.

3.2.5.1 Menus

The GNN has three menus: *GaRun, About, and Exit.* The functions of these menus are self-explanatory: *GaRun* starts program execution. *About* displays software information, and *Exit* allows the user to exit the program.

3.2.5.2 Network Parameter Information

This frame contains four subframes: Network Type, Number of Hidden [Units], Transfer Function, and Number of Layers. Network Type offers a choice of the three types of network supported in GNN: Back Propagation, Time Delayed Neural Network, and Continuous Adaptive Time Neural Network. Combined with other parameters, it forms a unique neural network topology for data training. Three numbers are displayed -- 3, 5, and 7 -- in Number of Hidden Units. The Transfer Function frame displays three functions: Logistic, Hyperbolic Tangent, and Linear Function. One of these transfer functions is implemented in the hidden layer(s). The Number of [Hidden] Layers frame is not implemented at this time; the number of hidden layers is fixed at one. The three variable parameters can generate 27 combinations (3×3×3). The goal of the software is to find the combination that minimizes the error rate.

3.2.5.3 GA Statistics

This frame contains two subframes, GA Progress Info[rmation] and Best Combination. The first row in the Progress Information box is for debugging, and is immaterial to the software. The text box in the second row displays the number of passes in the neural network training completed so far. Every time the neural network iterates, it adjusts the weights depending on the performance of the particular set of neurons and feeds back the result to the next iteration. Up to 1000 passes are implemented now. The last row displays the fitness value of the chromosome that contains the information on the neural network topology selected for data training: network type, number of hidden units, and transfer function used. In the GNN, since a neural network is used to approximate the fitness function, the fitness value of a genetic algorithm is the average error rate of the neural network.

The Best Combination frame displays information about the best neural network topology found so far in the current GNN session. It displays the network type, number of hidden units, type of transfer function, and the value of the fitness function.

3.2.5.4 Command Controls

The two controls buttons in the GNN, *Go GA* and *Exit*, have the same effect as the two menus, *GaRun* and *Exit*. When the user clicks on the *Go GA* button, one selection from each group in the Network Parameter Information frame turns cyan, showing that the colored component has been selected for constructing the neural network to be trained. The source code for GNN is listed in Appendix 1.

3.3 Mathlink Optimizer

Figure 3-4 shows the use of POC's GA Optimizer Mathematica plug-in for finding maximum values for a modified form of Himmelblau's function.

```
    MaxiMize2  ["200 - (x1^2 + x2 - 11)^2 - (x1 + x2^2 - 7)^2",
                          -10.0, 10.0, 0.01, -10.0, 10.0, 0.01]

Out [26]=
    {-2.80273, 3.134765625, 199.999324689125}
In [27]:=

    MaxiMize2  ["200 - (x1^2 + x2 - 11)^2 - (x1 + x2^2 - 7)^2",
                          -10.0, 10.0, 0.01, -10.0, 10.0, 0.01]

Out [27]=
    {2.998046875, 2.001953125, 199.999870330066}

    MaxiMize2  ["200 - (x1^2 + x2 - 11)^2 - (x1 + x2^2 - 7)^2",
                          -10.0, 10.0, 0.01, -10.0, 10.0, 0.01]

Out [28]+
    {-3.7793, -3.28125, 199.999835462295}
```

Figure 3-4
Sample run of MaxiMize for Himmelblau's function.

As shown in this figure, we can run Optimizer more than once to find multiple sets of argument values, if there are any for the same optima.

3.4 Function Optimizer Update

The function minimization program developed at the start of this project was revised, and the five most important functions were rewritten as Dynamic Link Libraries (DLLs), making them callable from any Windows application program. These functions are functionString, initializeMinimization, iterateMinimization, terminateMinimization, and getResults. This is another step toward developing general-purpose Application Program Interfaces (APIs). It shows that GA APIs can be used in many applications with just a slight code change. The source code is listed in Appendix 2.

3.4.1 Dynamic Link Library APIs

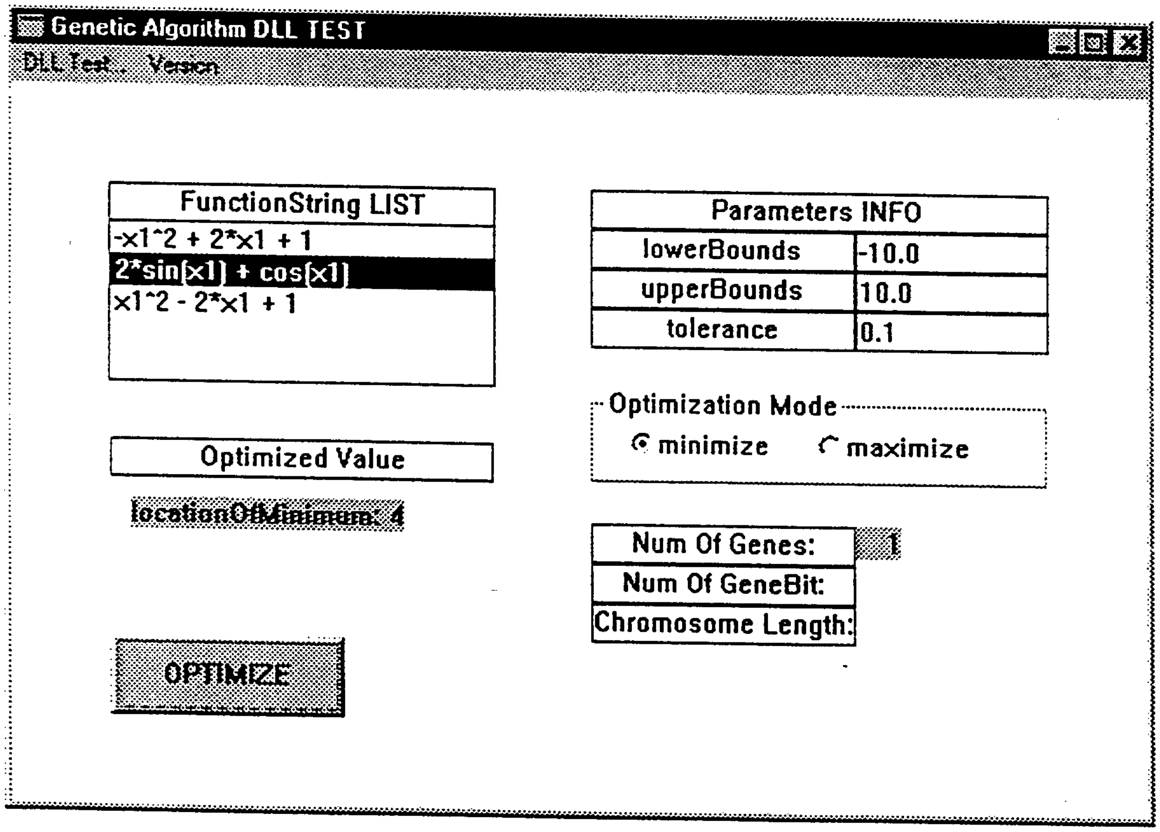
DLLs are Windows-based program modules that can be loaded and linked at run time. Since most non-trivial Windows programs are large because they include a graphical user interface as well as a programming interface, it is customary for programmers to develop APIs in DLL format, so that they can afford valuable functions to application developers while hiding code from the user. Of the five functions converted to DLL format, functionString parses the input function string and interprets it, and the others are self-explanatory. The algorithmic description using these functions is as follows:

```
functionString
initializeMinimization
for i = 1 to i < exit_value
    iterateMinimization
    i=i+1
end
terminateMinimization
getResults
```

To use these modules, all an application developer need do is to write a user interface and establish a data communication channel among modules.

3.4.2 Software update

POC developed a sample Windows interface program to demonstrate the feasibility of porting GA core functions to other applications. The GUI of the DLL test program is shown in Figure 3-5 and its source code is listed in Appendix 3.



FunctionString LIST
-x1^2 + 2*x1 + 1
2*sin[x1] + cos[x1]
x1^2 - 2*x1 + 1

Parameters INFO	
lowerBounds	-10.0
upperBounds	10.0
tolerance	0.1

Optimization Mode
[✓ minimize] maximize

Optimized Value
locationOfMinimum: 4

Num Of Genes:	1
Num Of GeneBit:	
Chromosome Length:	

Figure 3-5
Genetic algorithm DLL test program.

The user can select a function from the menu or type one in the *FunctionString LIST* box. The *functionString()* function picks up the string input, parses it, and builds a semantic tree. The range of variables can be given using the *Parameters INFO* box. Here the minimum and maximum boundary of each parameter can be adjusted. *Optimization Mode* allows the user to select either *minimization* or *maximization* mode. Either *initializeMinimization()* or *initializeMaximization()* will generate the initial values of chromosomes. At this point the user can select *OPTIMIZE* to start evolution; *iterateMinimization()* will continue evolution until the termination condition is met. The output information is reported in *Optimized Value. Num of Genes, Num of GeneBits*, and *Chromosome Length* are intended to help the developers understand the inner workings of the evolution process.

3.5 Building an Application on TI DSP/Genesis

The TI board hosts a 32-bit RISC master processor with an integrated floating point unit (FPU) capable of 100 MFLOPS, and four 32-bit parallel processors with a combined power of 2 billion operations per second. Its crossbar network has an on-chip data transfer rate of 2.4 Gbytes/second, significantly accelerating matrix calculation. To take full advantage of the DSP's processing power, we established a fast communication channel between the motherboard and the processor board, allowing us to take advantage of its memory resources, display, and input capabilities. Figure 3-6 diagrams the relationship between the application and the subsystems.

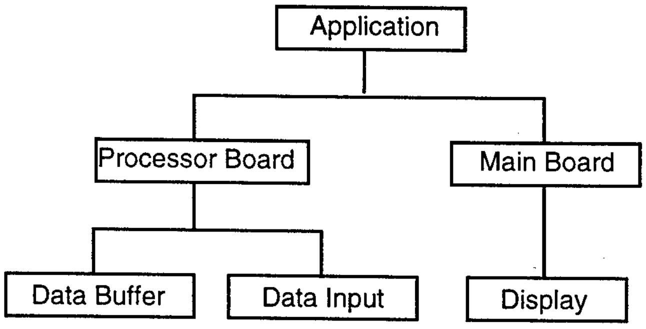


Figure 3-6
Relationship between application and hardware subsystems.

The program code that establishes a communication channel and allocates buffer memory on the processor board to the input is reproduced as Listing 3-1:

Listing 3-1 Establish Communication Channel and Allocate Buffer Memory.

```
...............
// Allocate an application
MappAlloc(M_DEFAULT,&MilApplication);
// Disable default Matrox Imaging Library (MIL) error message display
MappControl(M_ERROR,M_PRINT_DISABLE);
// Retrieve previous and user handler pointer
MappInquire(M_CURRENT_ERROR_HANDLER_PTR,&HandlerPtr);
MappInquire(M_CURRENT_ERROR_HANDLER_USER_PTR,&HandlerUserPtr);
// Hook MIL error on function DisplayError()
MappHookFunction(M_ERROR_CURRENT,DisplayErrorExt,this);
// Allocate a system
MsysAlloc(M_SYSTEM_SETUP,M_DEF_SYSTEM_NUM,M_COMPLETE,
&MilSystem);
................
```

After informing the host computer of the application we are running, we allocate the processor board system resources so that they can be used by the application. The search space is then loaded in the memory buffer on the processor board for further processing. The code in Listing 3-1 has been integrated into GA applications and a performance acceleration of 60% over the same CPU without the processor board was observed as shown in Table 3-2.

Table 3-2. Convergence Time Versus Size of the Problem.

Search Algorithm	Problem, Size N	Convergence Time in seconds for t = s and N = 10,000
Unordered Sequential	10,000	100
Ordered Sequential	10,000	13.29
GA w/o TI C80 DSP	10,000	4
GA with TI C80 DSP	10,000	0.2

This speedup is largely due to the C80's parallel processing capability. Some C80 processor board performance benchmarks are listed in Table 3-3.

Table 3-3. Processing Performance Benchmarks of TI TMS320C80.

Operation (512x512x8 images)		Processing Time with C80 (running at 50 MHz)
Histogram		3.0
Pattern Matching	128x128 model	10.0
	32x32 model	20.0
Convolution (with overflow saturation) 3x3		9.5
Image Rotation (bilinear interpolation)		22

4.0 CONCLUSIONS

In this Phase II, POC developed a GA-based decision making system of high dimensionality with high processing speed and robust optimization features. The proposed FEPGA technology has been demonstrated in a number of applications: Mathlink optimizer, Route optimizer, Genetic neural network, and Function optimizer. The genetic algorithm has also been implemented in a parallel computing environment, resulting in a considerable speedup. The above applications are just a few examples of how a genetic algorithm can be used.

The convergence speed of a genetic algorithm can be further improved by implementing it in a distributed computing environment, in which the computing task is divided into multiple subtasks with lower workloads. The genetic algorithm will be independently applied to subtasks for fast optimization, and the final result will be tallied through variables in shared memory. Such an implementation is suitable for problem domains such as aerial combat simulation or any commercial games involving multiple players and requiring fast optimization.

5.0 REFERENCES

1. G. Miller, P. Todd, S. Megde, "Designing Neural Networks Using Genetic Algorithms," ICGA '89. pp. 379-384.
2. S. Harp, T. Sumad, A. Guha, "Towards the Genetic Synthesis of Neural Networks," ICGA '89. pp. 360-369.
3. D.E. Rumhelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation," Parallel Distributed Processing: Explorations in the Microstructures of Cognition, D. E. Rumhelhart and J. L. McLelland, Eds., MIT Press, pp. 318-362 (1986).
4. X. Yao, "A Review of Evolutionary Artificial Neural Networks," Tech Rep. Commonwealth Scientific and Industrial Research Organization, Victoria (1992).
5. P. J. B. Hancock and L. S. Smith, "Gannet: Genetic Design of a Neural Network for Face Recognition," in Proc. Parallel Problem Solving from Nature, H. P. Schwefel and R. Manner, Eds. PPSN-1, Heidelberg: Springer Verlag, pp. 292-296 (1991).
6. D. Parisi, F. Cecconi, and S. Nolfi, "Econets: Neural Networks that Learn in an Environment," Network, 1, pp. 149-168 (1990).
7. D. Whitley, T. Starkweather, and C. Bogart, "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity," Computing, 14, pp. 347-361 (1990).

APPENDIX 1
GENETIC NEURAL NETWORK

```
' POC Genetic Neural Network
' Neural Network Parameter Optimization Program
' 12/96 - 10/97
' Physical Optics Corporation


Private Sub cmdExit_Click()
  End
End Sub

Private Sub cmdGo_Click()

  RunGeneticSearch

End Sub


Private Sub Form_Load()
  Dim Return_Code%
  'Main1.Show

  BestValue# = 1#
  Return_Code% = Initialize()
  Setup_Data
End Sub


Private Sub mnuAbout_Click()
  About.Show
End Sub

Private Sub mnuExit_Click()
  End
End Sub

Private Sub mnuGaRun_Click()
  RunGeneticSearch
End Sub
```

```
' POC NetCreator Main Module
' Nov 1996 - Jan 1997


Private Sub cmdDone_Click()
  'Free the network from memory
  Ret% = Release_Network(Net_ID%)
  Net_ID% = -1  'to show that it is no longer valid

  Unload Me
  End
End Sub

Private Sub cmdRedo_Click()
  Dim Return_Code%

  Main1.txtError.Text = "          "
  Main1.txtNumPass.Text = "         "

  ' reset the data display
  For Record_Nbr% = 1 To Nbr_Records% + 1
      For Column_Nbr% = 1 To 4     ' Needs change later
        Main1.gridStats.Row = Record_Nbr%
        Main1.gridStats.Col = Column_Nbr%
        Main1.gridStats.Text = "     "
      Next Column_Nbr%
  Next Record_Nbr%

  'Free the network from memory
  Ret% = Release_Network(Net_ID%)
  Net_ID% = -1  'to show that it is no longer valid

  Return_Code% = Initialize()
  Setup_Data
End Sub


Private Sub cmdTrain_Click()
  Build_Network
  Train_Network
End Sub


Private Sub Form_Load()
  Dim Return_Code%

  Return_Code% = Initialize()
  Setup_Data

  ' set the row and column size
  For Record_Nbr% = 0 To Nbr_Records% + 1
      For Column_Nbr% = 1 To 4     ' Needs change later
        gridStats.RowHeight(Record_Nbr%) = 460
        gridStats.ColWidth(Column_Nbr%) = 820
      Next Column_Nbr%
  Next Record_Nbr%

  gridStats.RowHeight(6) = 570
  gridStats.ColWidth(0) = 1000

  ' display the input number
  For Record_Nbr% = 1 To Nbr_Records%
        Main1.gridStats.Row = Record_Nbr%
        Main1.gridStats.Col = 0
        Main1.gridStats.Text = Record_Nbr%
  Next Record_Nbr%
```

```
    ' display captions
    Main1.gridStats.Row = Record_Nbr%
    Main1.gridStats.Col = 0
    Main1.gridStats.Text = "   Average        Error Rate"

    Main1.gridStats.Row = 6
    Main1.gridStats.Col = 0
    Main1.gridStats.Text = "    TEST              INPUT"

    Main1.gridStats.Row = 0
    Main1.gridStats.Col = 1
    Main1.gridStats.Text = "1st Variable"

    Main1.gridStats.Row = 0
    Main1.gridStats.Col = 2
    Main1.gridStats.Text = "2nd Variable"

    Main1.gridStats.Row = 0
    Main1.gridStats.Col = 3
    Main1.gridStats.Text = "Output"

    Main1.gridStats.Row = 0
    Main1.gridStats.Col = 4
    Main1.gridStats.Text = "Neural Output"

End Sub


Private Sub mnuFileItem_Click(Index As Integer)
  Select Case Index
    Case 0
      Unload Me
      End
    Case 2
      Main1.PrintForm
  End Select
End Sub

Private Sub SSOption1_Click()
  Net_Type% = BP
End Sub

Private Sub SSOption2_Click()
  Net_Type% = TDNN
End Sub

Private Sub SSOption3_Click()
  Net_Type% = CATNN
End Sub

Private Sub SSOption4_Click()
  Net_Type% = PNN
End Sub

Private Sub SSOption5_Click()
  Net_Type% = LVQ
End Sub

Private Sub tlbPrint_Click()
   Main1.PrintForm
End Sub
```

```
Option Explicit

'     '============================================================
'     'Genetic Algorithm Dynamic Link Library Declarations
'     '------------------------------------------------------------
    Declare Function InitBinary Lib "GENE200.DLL" (ByVal StartingID%, ByVal EndingID%, ByVal Peg
gedID%, ByVal HighLow%, ByVal StrandLen%, ChromeGene%, ChromeValue#) As Integer
    Declare Function InitInt Lib "GENE200.DLL" (ByVal StartingID%, ByVal EndingID%, ByVal StartV
al%, ByVal EndVal%, ByVal Unique%, ByVal PeggedGene1%, ByVal Gene1Val%, ByVal StrandLen%, Chrome
Gene%, ChromeValue#) As Integer
    Declare Function InitZero Lib "GENE200.DLL" (ByVal PopSize%, ByVal StrandLen%, ChromeGene%,
ChromeValue#) As Integer
    Declare Function SelectPercent Lib "GENE200.DLL" (ByVal Percent#, ByVal HighLow%, ByVal PopS
ize%, ChromeValue#, SurvivorList%) As Integer
    Declare Function SelectRoulette Lib "GENE200.DLL" (ByVal RoulOrder%, ByVal HighLow%, ByVal P
opSize%, ChromeValue#, SurvivorList%) As Integer
    Declare Function RefillBinaryRand Lib "GENE200.DLL" (ByVal NumberOfSurvivors%, ByVal PopSize
%, ByVal StrandLen%, ChromeGene%, ChromeValue#, SurvivorList%) As Integer
    Declare Function RefillClone Lib "GENE200.DLL" (ByVal NumberOfSurvivors%, ByVal PopSize%, By
Val StrandLen%, ChromeGene%, ChromeValue#, SurvivorList%) As Integer
    Declare Function PairRandom Lib "GENE200.DLL" (ByVal PopSize%, ParentPair%) As Integer
    Declare Function MateTailSwap Lib "GENE200.DLL" (ByVal StrandLen%, ByVal NumPairs%, ChromeGe
ne%, ParentPair%) As Integer
    Declare Function MateTwoCut Lib "GENE200.DLL" (ByVal PopSize%, ByVal StrandLen%, ByVal NumPa
irs%, ChromeGene%, ParentPair%) As Integer
    Declare Function MateTwoCutSwap Lib "GENE200.DLL" (ByVal PopSize%, ByVal StrandLen%, ByVal N
umPairs%, ChromeGene%(), ParentPair%) As Integer
    Declare Function MuteRandEx Lib "GENE200.DLL" (ByVal RandExRate#, ByVal PreserveGeneOne%, By
Val PopSize%, ByVal StrandLen%, ChromeGene%) As Integer
    Declare Function MuteRev Lib "GENE200.DLL" (ByVal RandRevRate#, ByVal PreserveGeneOne%, ByVa
l PopSize%, ByVal StrandLen%, ChromeGene%) As Integer
    Declare Function Rand_List Lib "GENE200.DLL" (ByVal Nbr_Items%, Index_List%()) As Integer

'=======================================================
'These array's are dimensioned to 50 to allow room for more
'cities if you like.
Global DistanceArray(50, 50) As Integer
Global IndexList(50) As Integer
Global SequencedList(50) As Integer
Global BestChrome(50) As Integer

Global BestValue As Double
Global FitnessValue As Double
Global Net_Error As Double
Global BestType As Integer
Global BestHiddenNum As Integer
Global BestFunction As Integer

'NOTE: IMPORTANT
'These arrays, with GENE200.DLL MUST be dimensioned in this manner
Global ChromeGene%(0 To 174, 0 To 174)
Global ChromeValue#(0 To 174)
Global SurvivorList%(0 To 174)
Global ParentPair%(0 To 174)

'Misc other global variables
Global NotFirstPass As Integer
Global HighLow As Integer
Global Const Nbr_Selected_Parameters = 3

'Newly added global variables: Genetic Parameters
Global Net_Type As Integer
Global Nbr_Hiddens(3) As Integer
Global Transfer_Function(3) As Integer
```

```
Option Explicit

#If Win16 Then
    Declare Function Initialize Lib "NNW16212.dll" () As Integer
    Declare Function Build_BP Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Inputs
As Integer, ByVal Nbr_Hidden_Layers As Integer, Nbr_Hiddens As Integer, Transfer_Function As Int
eger, ByVal Nbr_Outputs As Integer) As Integer
    Declare Function Build_AT Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Inputs
As Integer, ByVal Nbr_Hidden_Layers As Integer, Nbr_Hiddens As Integer, Transfer_Function As Int
eger, Nbr_Connections As Integer, ByVal Max_Tau As Integer, ByVal Nbr_Outputs As Integer) As Int
eger
    Declare Function Build_PNN Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Inputs
 As Integer, ByVal Nbr_of_Records As Integer, ByVal Nbr_Outputs As Integer) As Integer
    Declare Function Build_GRNN Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Input
s As Integer, ByVal Nbr_of_Records As Integer, ByVal Nbr_Outputs As Integer) As Integer
    Declare Function Init_Weights Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Lower_L
imit#, ByVal Upper_Limit#) As Integer
    Declare Function Init_Taus Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Lower_Limi
t#, ByVal Upper_Limit#) As Integer
    Declare Function Propagate_BP Lib "NNW16212.dll" (ByVal Network_ID As Integer, Input_Array#,
 Desired_Output_Array#) As Integer
    Declare Function Propagate_AT Lib "NNW16212.dll" (ByVal Network_ID As Integer, Input_Array#,
 Desired_Output_Array#) As Integer
    Declare Function Propagate_PNN Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal sigma#
, Input_Array#, Output_Array#) As Integer
    Declare Function Propagate_GRNN Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal sigma
#, Input_Array#, Output_Array#) As Integer
    Declare Function Calc_Net_Error Lib "NNW16212.dll" (ByVal Network_ID As Integer, Desired_Out
put_Array#) As Double
    Declare Function Train_BP Lib "NNW16212.dll" (ByVal Network_ID As Integer, Learn_Rate#, Mome
ntum#) As Integer
    Declare Function Train_AT Lib "NNW16212.dll" (ByVal Network_ID As Integer, Learn_Rate#, Mome
ntum#, Tau_Learn_Rate#, Tau_Momentum#, ByVal Commit_Changes As Integer) As Double

    Declare Function Train_PNN Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Record_ID%
, Input_Array#, Output_Array#) As Integer
    Declare Function Train_GRNN Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal Record_ID
%, Input_Array#, Output_Array#) As Integer

    Declare Function Release_Network Lib "NNW16212.dll" (ByVal Network_ID As Integer) As Integer
    Declare Function Release_All_Networks Lib "NNW16212.dll" () As Integer
    Declare Function Save_Net Lib "NNW16212.dll" (ByVal FileName$, ByVal Network_ID As Integer)
As Integer
    Declare Function Load_Net Lib "NNW16212.dll" (ByVal FileName$, ByVal Network_ID As Integer)
As Integer
    Declare Function MoveNets Lib "NNW16212.dll" (ByVal Old_Index As Integer, ByVal New_Index As
 Integer) As Integer
    Declare Function GetNetID Lib "NNW16212.dll" () As Integer
    Declare Function SetParameters Lib "NNW16212.dll" (ByVal Network_ID As Integer, ByVal log_ga
in#, ByVal log_mag#, ByVal tanh_gain#, ByVal tanh_mag#, ByVal lin_slope#) As Integer


#ElseIf Win32 Then
    Declare Function Initialize Lib "NNW32212.dll" () As Integer
    Declare Function Build_BP Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Inputs
As Integer, ByVal Nbr_Hidden_Layers As Integer, Nbr_Hiddens As Integer, Transfer_Function As Int
eger, ByVal Nbr_Outputs As Integer) As Integer
    Declare Function Build_AT Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Inputs
As Integer, ByVal Nbr_Hidden_Layers As Integer, Nbr_Hiddens As Integer, Transfer_Function As Int
eger, Nbr_Connections As Integer, ByVal Max_Tau As Integer, ByVal Nbr_Outputs As Integer) As Int
eger
    Declare Function Build_PNN Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Inputs
 As Integer, ByVal Nbr_of_Records As Integer, ByVal Nbr_Outputs As Integer) As Integer
    Declare Function Build_GRNN Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Nbr_Input
s As Integer, ByVal Nbr_of_Records As Integer, ByVal Nbr_Outputs As Integer) As Integer
    Declare Function Init_Weights Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Lower_L
imit#, ByVal Upper_Limit#) As Integer
    Declare Function Init_Taus Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Lower_Limi
t#, ByVal Upper_Limit#) As Integer
```

```
    Declare Function Propagate_BP Lib "NNW32212.dll" (ByVal Network_ID As Integer, Input_Array#,
 Desired_Output_Array#) As Integer
    Declare Function Propagate_AT Lib "NNW32212.dll" (ByVal Network_ID As Integer, Input_Array#,
 Desired_Output_Array#) As Integer
    Declare Function Propagate_PNN Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal sigma#
, Input_Array#, Output_Array#) As Integer
    Declare Function Propagate_GRNN Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal sigma
#, Input_Array#, Output_Array#) As Integer
    Declare Function Calc_Net_Error Lib "NNW32212.dll" (ByVal Network_ID As Integer, Desired_Out
put_Array#) As Double
    Declare Function Train_BP Lib "NNW32212.dll" (ByVal Network_ID As Integer, Learn_Rate#, Mome
ntum#) As Integer
    Declare Function Train_AT Lib "NNW32212.dll" (ByVal Network_ID As Integer, Learn_Rate#, Mome
ntum#, Tau_Learn_Rate#, Tau_Momentum#, ByVal Commit_Changes As Integer) As Double
    Declare Function Train_PNN Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Record_ID%
, Input_Array#, Output_Array#) As Integer
    Declare Function Train_GRNN Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal Record_ID
%, Input_Array#, Output_Array#) As Integer
    Declare Function Release_Network Lib "NNW32212.dll" (ByVal Network_ID As Integer) As Integer
    Declare Function Release_All_Networks Lib "NNW32212.dll" () As Integer
    Declare Function Save_Net Lib "NNW32212.dll" (ByVal FileName$, ByVal Network_ID As Integer)
As Integer
    Declare Function Load_Net Lib "NNW32212.dll" (ByVal FileName$, ByVal Network_ID As Integer)
As Integer
    Declare Function MoveNets Lib "NNW32212.dll" (ByVal Old_Index As Integer, ByVal New_Index As
 Integer) As Integer
    Declare Function GetNetID Lib "NNW32212.dll" () As Integer
    Declare Function SetParameters Lib "NNW32212.dll" (ByVal Network_ID As Integer, ByVal log_ga
in#, ByVal log_mag#, ByVal tanh_gain#, ByVal tanh_mag#, ByVal lin_slope#) As Integer
#End If
```

```
Option Explicit

Global Net_ID As Integer      'a unique identifier for our neural network
'Global Net_Type As Integer    'The type of network we're building (BP, TDNN, ...)
Global Nbr_Records As Integer

'Constants to make things clearer in code
'Network Types
Global Const BP = 0
Global Const TDNN = 1
Global Const CATNN = 2
Global Const PNN = 3
Global Const LVQ = 4

'Transfer functions
Global Const LOGISTIC = 0
Global Const TANH = 1
Global Const LINEAR = 2

'Network Architecture Variables

'Data Arrays
Global DataArray() As Double  'An array to hold our data
                              'We will dimension it in Build_Network function
Global Test_Array#(2)       'an array to hold a record of test input data
Global Test_Output_Array#(1)

Sub Build_Network()

    Dim Nbr_Inputs%
    Dim Nbr_Hidden_Layers%
    'ReDim Nbr_Hiddens%(3)
    'ReDim Transfer_Function%(3)
    ReDim Nbr_Connections%(3)
    Dim Max_Tau%
    Dim Nbr_Outputs%
    Dim Ret%

    'Let's set some network parameters...
    Nbr_Inputs% = 2
    Nbr_Hidden_Layers% = 1
    'Nbr_Hiddens%(1) = 5
    'Nbr_Hiddens%(2) = 5
    Transfer_Function%(1) = TANH
    'Transfer_Function%(2) = LOGISTIC
    Nbr_Connections%(1) = 1
    Nbr_Connections%(2) = 1
    Max_Tau% = 10
    Nbr_Outputs% = 1

    Net_ID% = GetNetID()


    If Net_ID% < 0 Then
        MsgBox "Error getting network ID", 48, "POC Network Generator"
    End If

    Select Case Net_Type%
        Case BP
            Ret% = Build_BP(Net_ID%, Nbr_Inputs%, Nbr_Hidden_Layers%, Nbr_Hiddens%(0), Transfer_
 Function%(0), Nbr_Outputs%)
        Case TDNN
            Ret% = Build_AT(Net_ID%, Nbr_Inputs%, Nbr_Hidden_Layers%, Nbr_Hiddens%(0), Transfer_
 Function%(0), Nbr_Connections%(0), Max_Tau, Nbr_Outputs%)
        Case CATNN
            Ret% = Build_AT(Net_ID%, Nbr_Inputs%, Nbr_Hidden_Layers%, Nbr_Hiddens%(0), Transfer_
 Function%(0), Nbr_Connections%(0), Max_Tau, Nbr_Outputs%)
        Case PNN
            Ret% = Build_PNN(Net_ID%, Nbr_Inputs%, Nbr_Records%, Nbr_Outputs%)
        Case LVQ
            Ret% = Build_GRNN(Net_ID%, Nbr_Inputs%, Nbr_Records%, Nbr_Outputs%)
    End Select
    If Ret% < 0 Then
        MsgBox "Error building network", 48, "POC Network Generator"
    End If

End Sub


Sub Setup_Data()
    'Let's setup our data here, you may wish to use your own functions
    'to do this when building applications
    ReDim DataArray#(4, 3)  '4 rows, 3 columns (2 inputs, 1 output)
    Nbr_Records% = 4

    'Let's put xor data in the array
    DataArray#(1, 1) = 1
    DataArray#(1, 2) = 0
    DataArray#(1, 3) = 1

    DataArray#(2, 1) = 0
    DataArray#(2, 2) = 1
    DataArray#(2, 3) = 1

    DataArray#(3, 1) = 0
    DataArray#(3, 2) = 0
    DataArray#(3, 3) = 0

    DataArray#(4, 1) = 1
    DataArray#(4, 2) = 1
    DataArray#(4, 3) = 0

End Sub

Sub Train_Network()

    Dim Ret%                      'A variable to hold return codes
    Dim Passes As Long            'The number of passes through the data
    Dim Record_Nbr%               'A simple index of records
    Dim Column_Nbr%               'A simple index of columns
    Dim error_fact#               'The mean squared error of BP/TDNN/CATNN network
    'Dim Net_Error#                'The Average Absolute Error of the network
    Dim Commit_Changes%           'Whether to update weights this pass
    Dim sigma#                    'The acuity factor for PNN's and LVQ's
    Dim cum_error#                 'An accumulation of error across all records

    ReDim Learn_Rate#(3)          'The network learning rate
    ReDim Momentum#(3)            'The network momentum rate
    ReDim Tau_Learn_Rate#(3)      'CATNN connection 'look back' learning rate
    ReDim Tau_Momentum#(3)        'CATNN connection 'look back' momentum

    ReDim Input_Array#(2)         'an array to hold a record of input data
    ReDim Output_Array#(2)        'an array to hold a record of Output data
    ReDim Neural_Output_Array#(1)   'an array to hold the neural results

    Learn_Rate#(1) = 0.8          'set hidden layer learning rate
    Learn_Rate#(2) = 0.4          'set output layer learning rate
    Momentum#(1) = 0.2            'set hidden layer Momentum
    Momentum#(2) = 0.1            'set output layer Momentum
    Commit_Changes% = 1           'We're not doing epoch based learning, so always update weights

    Tau_Learn_Rate#(1) = 0        'We're not updating CATNN connections, as this isn't a time base
d problem
    Tau_Learn_Rate#(2) = 0
    Tau_Momentum#(1) = 0
    Tau_Momentum#(2) = 0
    error_fact# = 1000            'some big value to start
    Net_Error# = 1000             'some big value to start
    Passes& = 0
    sigma# = 0.01

    'Initialize Weights
    Ret% = Init_Weights(Net_ID, -0.3, 0.3)
    If Net_Type% = CATNN Or Net_Type% = TDNN Then
        Ret% = Init_Taus(Net_ID, -0.3, 0.3)
    End If

    ' display of input and output
    For Record_Nbr% = 1 To Nbr_Records%
      For Column_Nbr% = 1 To 3    ' Needs change later
        Main1.gridStats.Row = Record_Nbr%
        Main1.gridStats.Col = Column_Nbr%
        Main1.gridStats.Text = Format$(DataArray#(Record_Nbr%, Column_Nbr%), "0.0000")
      Next Column_Nbr%
    Next Record_Nbr%

    Do While (Net_Error# > 0.001 And Passes& < 1000)   'train until the error meets a criteria
        DoEvents
        Passes& = Passes& + 1
        cum_error# = 0
        For Record_Nbr% = 1 To Nbr_Records%

            'Load the current record into the Input and Output arrays
            Input_Array#(1) = DataArray#(Record_Nbr%, 1)
            Input_Array#(2) = DataArray#(Record_Nbr%, 2)
            Output_Array#(1) = DataArray#(Record_Nbr%, 3)

            'Propagate forward
            Select Case Net_Type%
                Case BP
                    Ret% = Propagate_BP(Net_ID%, Input_Array#(0), Neural_Output_Array#(0))
                Case CATNN, TDNN
                    Ret% = Propagate_AT(Net_ID%, Input_Array#(0), Neural_Output_Array#(0))
                Case PNN, LVQ
                    'no need to propagate PNN, LVQ first during training
            End Select
            If Ret% <> 0 Then
                MsgBox "Error Propagating in NNWIN.DLL", 48, "POC Network Generator"
            End If

            'Calculate error
            Select Case Net_Type%
                Case BP, CATNN, TDNN
                    error_fact# = Calc_Net_Error(Net_ID%, Output_Array#(0))
                    If error_fact# < 0 Then
                        MsgBox "Error Calculating Network Error in NNWIN.DLL", 48, "POC Network
 Generator"
                    End If
                Case Else
                    'no need to calc net error for PNN/LVQ during training
            End Select

            Select Case Net_Type%
                Case BP
                    Ret% = Train_BP(Net_ID%, Learn_Rate#(0), Momentum#(0))
                Case TDNN, CATNN
                    Ret% = Train_AT(Net_ID%, Learn_Rate#(0), Momentum#(0), Tau_Learn_Rate#(0), T
 au_Momentum#(0), Commit_Changes%)
                Case PNN
                    Ret% = Train_PNN(Net_ID%, Record_Nbr%, Input_Array#(0), Output_Array#(0))
                Case LVQ
                    Ret% = Train_GRNN(Net_ID%, Record_Nbr%, Input_Array#(0), Output_Array#(0))
            End Select
            If Ret% < 0 Then
                MsgBox "Error building network", 48, "POC Network Generator"
            End If
            'Propagate forward to get the network's prediction
            Select Case Net_Type%
                Case BP
                    Ret% = Propagate_BP(Net_ID%, Input_Array#(0), Neural_Output_Array#(0))
                Case CATNN, TDNN
                    Ret% = Propagate_AT(Net_ID%, Input_Array#(0), Neural_Output_Array#(0))
                Case PNN
                    Ret% = Propagate_PNN(Net_ID%, sigma#, Input_Array#(0), Neural_Output_Array#(
0))
                Case LVQ
                    Ret% = Propagate_GRNN(Net_ID%, sigma#, Input_Array#(0), Neural_Output_Array#
(0))
            End Select
            If Ret% <> 0 Then
                MsgBox "Error Propagating in NNWIN.DLL", 48, "POC Network Generator"
            End If

            'Calculate error
            cum_error# = cum_error# + Abs(Output_Array#(1) - Neural_Output_Array#(1))

            'Main1->GNN
            'NeuroGen.gridStats.Row = Record_Nbr%
            'NeuroGen.gridStats.Col = 4
            'NeuroGen.gridStats.Text = Format$(Neural_Output_Array#(1), "0.0000")

        Next Record_Nbr%

        Net_Error# = cum_error# / Nbr_Records%

        'Show resulting error and the number of pass
        NeuroGen.txtError.Text = Format$(Net_Error#, "0.0000")
        NeuroGen.txtNumPass.Text = Str$(Passes&)

        'NeuroGen.gridStats.Row = 5
        'NeuroGen.gridStats.Col = 4
        'NeuroGen.gridStats.Text = Format$(Net_Error#, "0.0000")
    Loop

End Sub
```

```
Function CustomMutation(ByVal PreserveGeneOne%, ByVal PopSize%, ByVal StrandLength%)

    'This is an example of a custom function that you can
    'build to compliment the GAWindows Library, customizing
    'for special cases and for "hybrid" applications

    'This mutation reverses pieces of chromosomes using "knowledge"
    'of distances ("domain knowledge")

    'Find the two highest intercity distances in each chromosome
    'Mutate every chromosome because this method is so effective!

    Static GeneSegment%(50)
    Static RevGeneSeq%(50)


    For i% = 1 To PopSize%
        'set the highest and second highest distances to the first one
        'just to have something to compare against
        HighestDist# = DistanceArray%(IndexList%(ChromeGene%(i%, 1) - 1), IndexList%(ChromeGene%
(i%, 2) - 1))
        SecondHighestDist# = DistanceArray%(IndexList%(ChromeGene%(i%, 1) - 1), IndexList%(Chrom
eGene%(i%, 2) - 1))

        For j% = 1 To StrandLength% - 1

            'If highest distance is smaller than this distance
            If HighestDist# < DistanceArray%(IndexList%(ChromeGene%(i%, j%) - 1), IndexList%(Chr
omeGene%(i%, j% + 1) - 1)) Then
                'then the current highest distance is now the second highest
                SecondHighestDist# = HighestDist#
                'and the first cut point in the chromosome is now the second
                CutPoint2 = CutPoint1
                'and the new highest distance is this distance
                HighestDist# = DistanceArray%(IndexList%(ChromeGene%(i%, j%) - 1), IndexList%(Ch
romeGene%(i%, j% + 1) - 1))
                'and this is the new highest cut point
                CutPoint1 = j%
            End If

            If (SecondHighestDist# < DistanceArray%(IndexList%(ChromeGene%(i%, j%) - 1), IndexLi
st%(ChromeGene%(i%, j% + 1) - 1)) And (HighestDist# <> DistanceArray%(IndexList%(ChromeGene%(i%,
 j%) - 1), IndexList%(ChromeGene%(i%, j% + 1) - 1)))) Then
                SecondHighestDist# = DistanceArray%(IndexList%(ChromeGene%(i%, j%) - 1), IndexLi
st%(ChromeGene%(i%, j% + 1) - 1))
                CutPoint2 = j%
            End If

        Next j%

        'Found the Highest and SecondHighest Intercity distances
        'Now order the cutpoints along the chromosome
        If CutPoint1 <= CutPoint2 Then
            CutOne = CutPoint1
            CutTwo = CutPoint2
        Else
            CutOne = CutPoint2
            CutTwo = CutPoint1
        End If

        'Increment CutOne so as to "cut" on next boundary
        CutOne = CutOne + 1

        'Check for preservation of Gene 1
        If (CutOne = 1) And (PreserveGeneOne = 1) Then
            CutOne = 2
        End If
        'Sometimes reverse to end of string just for fun
        If Rnd > 0.5 Then
            CutTwo = StrandLength%
        End If

        'Now extract the genes
        For j% = CutOne To CutTwo
            GeneSegment%(j% - CutOne) = ChromeGene(i%, j%)
        Next j%

        'Now reverse it
        For j% = 0 To (CutTwo - CutOne)
            RevGeneSeq%(j%) = GeneSegment%(CutTwo - CutOne - j%)
        Next j%

        'Now Stick it Back
        For j% = CutOne To CutTwo
            ChromeGene%(i%, j%) = RevGeneSeq%(j% - CutOne)
        Next j%

    Next i%

End Function

Function Fitness(PopSize As Integer, StrandLength As Integer)

    'Determine miles travelled for each chromosome in population

    For i% = 1 To PopSize%  'for each chrome in population

        'Calculate Total Miles (Chromosome's "Value")
        ChromeValue#(i%) = 0    'zero out the value of the chromosome
        For j% = 1 To StrandLength% - 1

            'accumulate the distances
            ChromeValue#(i%) = ChromeValue#(i%) + DistanceArray%(IndexList%(ChromeGene%(i%, j%)
- 1), IndexList%(ChromeGene%(i%, j% + 1) - 1))

        Next j%

    Next i%

    'Look for lowest path so far
    If NotFirstPass% = 0 Then
        BestValue# = ChromeValue#(1)
        For j% = 1 To StrandLength%
            SequencedList%(j%) = ChromeGene%(1, j%)
        Next j%
        NotFirstPass% = 1
    End If

    For i% = 1 To PopSize%
        'if we're looking for the lowest and this chrome is lower than the lowest
        'or if we're looking for the highest and this chrome is higher than the highest
        If (HighLow = 0 And ChromeValue#(i%) < BestValue#) Or (HighLow = 1 And ChromeValue#(i%)
> BestValue#) Then
            'Take this chromosome's distance as best
            BestValue# = ChromeValue#(i%)
            'and copy the chromosome to the desired sequence of cities to travel
            For j% = 1 To StrandLength%
                SequencedList%(j%) = ChromeGene%(i%, j%)
            Next j%
        End If
    Next i%

    'return the smallest distance
    Fitness = BestValue#

End Function
```

```
Function RunGeneticSearch() As Integer
    'Show Hourglass 'cause we're busy!
    NeuroGen.MousePointer = 11  'EDITED

    'Set Genetic parameters
    PopSize% = Nbr_Selected_Parameters% * 4  'A good rule of thumb for this problem
    StrandLength% = Nbr_Selected_Parameters%
    GenerationLimit% = Nbr_Selected_Parameters% * 4 'another rule of thumb for TSP
    StartingID% = 1
    EndingID% = PopSize%
    StartVal% = 1
    EndVal% = StrandLength%
    Unique% = 0
    PeggedGeneOne% = 0
    GeneOneValue% = 0
    Percent# = 0.5
    RandExRate# = 0.5
    RandRevRate# = 0.5
    PreserveGeneOne% = 1

    'A flag for determining if this is the first time through
    NotFirstPass% = 0

    'Make the world topsy turvey (scramble the random #'s)
    Randomize

    'Now build the population of integers of City ID's
    Result% = InitInt%(1, PopSize%, 1, StrandLength%, Unique%, PeggedGeneOne%, GeneOneValue%, St
randLength%, ChromeGene%(0, 0), ChromeValue#(0))

    'Start the Genetic LifeCycle here
    Do While (Generations% < 1) 'GenerationLimit%)
        'Take Care of Business elsewhere in Windows
        Result% = DoEvents()

        For i% = 1 To PopSize%
           'Decode chromosome
           Net_Type% = ChromeGene%(i%, 1) - 1
           Select Case Net_Type%
             Case 0
               NeuroGen.txtNetType1.BackColor = &HFFFF00
             Case 1
               NeuroGen.txtNetType2.BackColor = &HFFFF00
             Case 2
               NeuroGen.txtNetType3.BackColor = &HFFFF00
           End Select

           Select Case ChromeGene%(i%, 2)
             Case 1
               Nbr_Hiddens%(1) = 3
               NeuroGen.txtNumHidden1.BackColor = &HFFFF00
             Case 2
               Nbr_Hiddens%(1) = 5
               NeuroGen.txtNumHidden2.BackColor = &HFFFF00
             Case 3
               Nbr_Hiddens%(1) = 7
               NeuroGen.txtNumHidden3.BackColor = &HFFFF00
           End Select

           Transfer_Function%(2) = ChromeGene%(i%, 3) - 1
           Select Case Transfer_Function%(2)
             Case 0
               NeuroGen.txtFunction1.BackColor = &HFFFF00
             Case 1
               NeuroGen.txtFunction2.BackColor = &HFFFF00
             Case 2
               NeuroGen.txtFunction3.BackColor = &HFFFF00
           End Select
```

```
          'TEST
           NeuroGen.txtDebug1.Text = Str$(Net_Type%)
           NeuroGen.txtDebug2.Text = Str$(Nbr_Hiddens%(1))
           NeuroGen.txtDebug3.Text = Str$(Transfer_Function%(2))

           Build_Network
           Train_Network

          'Evaluate Fitness
          'BestValue# = Fitness(PopSize%, StrandLength%)
          If Net_Error# < BestValue# Then
            BestValue# = Net_Error#
            BestType% = Net_Type%
            BestHiddenNum% = Nbr_Hiddens%(1)
            BestFunction% = Transfer_Function%(2)
          End If

          Select Case BestType%
            Case 0
              NeuroGen.txtBestType.Text = "Back Propagation"
            Case 1
              NeuroGen.txtBestType.Text = "Time Delay"
            Case 2
              NeuroGen.txtBestType.Text = "Continuous Adaptive"
          End Select

          'NeuroGen.txtBestType.Text = Format$(BestType%, "0.0000")

          NeuroGen.txtBestNum.Text = BestHiddenNum%

          'NeuroGen.txtBestNum.Text = Format$(BestHiddenNum%, "0.0000")

          Select Case BestFunction%
            Case 0
              NeuroGen.txtBestFunction.Text = "Logistic Function"
            Case 1
              NeuroGen.txtBestFunction.Text = "Hyperbolic Tangent"
            Case 2
              NeuroGen.txtBestFunction.Text = "Linear Transfer"
          End Select

          'NeuroGen.txtBestFunction.Text = Format$(BestFunction%, "0.0000")

          NeuroGen.txtBestRate.Text = Format$(BestValue#, "0.0000")

          NeuroGen.txtNetType1.BackColor = &HFFFFFF
          NeuroGen.txtNetType2.BackColor = &HFFFFFF
          NeuroGen.txtNetType3.BackColor = &HFFFFFF
          NeuroGen.txtNumHidden1.BackColor = &HFFFFFF
          NeuroGen.txtNumHidden2.BackColor = &HFFFFFF
          NeuroGen.txtNumHidden3.BackColor = &HFFFFFF
          NeuroGen.txtFunction1.BackColor = &HFFFFFF
          NeuroGen.txtFunction2.BackColor = &HFFFFFF
          NeuroGen.txtFunction3.BackColor = &HFFFFFF

       Next i%

       'Select survivors
           Nbr_Survivors% = SelectPercent%(Percent#, HighLow%, PopSize%, ChromeValue#(0), Survi
vorList%(0))

       'Refill Population
           Nbr_Chromes_Created% = RefillClone%(Nbr_Survivors%, PopSize%, StrandLength%, ChromeG
ene%(0, 0), ChromeValue#(0), SurvivorList%(0))

       'Pair for Mating
           Nbr_Pairs% = PairRandom%(PopSize%, ParentPair(0))
```

```
        'Exchange Genes
            Nbr_Matings% = MateTwoCut%(PopSize%, StrandLength%, Nbr_Pairs%, ChromeGene%(0, 0), P
arentPair%(0))

        'Mutate
            Nbr_Mutations% = MuteRev%(RandRevRate#, PreserveGeneOne%, PopSize%, StrandLength%, C
hromeGene%(0, 0))
            Nbr_Mutations% = MuteRandEx%(RandExRate#, PreserveGeneOne%, PopSize%, StrandLength%,
 ChromeGene%(0, 0))
            Nbr_Mutations% = CustomMutation(PreserveGeneOne%, PopSize%, StrandLength%)

        'Go back to evaluate
        Generations% = Generations% + 1
    Loop

    'Return best intercity distance found
    RunGeneticSearch = BestValue#

    'Put the mouse back to a pointer
    NeuroGen.MousePointer = 1

End Function
```

APPENDIX 2
DLL FOR MINIMIZATION GENETIC ALGORITHM

```
/*****************************************************************
    DLL for minimization genetic algorithm
******************************************************************/

/*
   Last modified by J. Kim 6/97
   This version is a callable function minimize(f(),...)
   This version is optimized for low memory use:
      bits are packed instead of one per byte;
      storage is dynamically allocated as needed
*/



#include<Float.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
#include <windows.h>

#include "minimize.h"
#include "crash.h"

// int initFKeyin( char *);

int n,i,j,t,k,tt,pp,ngene,slength,geneval,nbit[11], rep = 0, cross;
int p,int_rand,itt_rand,intm_pxval,it2_rand,intn,nb,nbold,ne,neold;
double maximum2,minimum2,maximum,minimum, mold, minold,mold2,minold2;
double gendec, diffmax, diffmin,fac;
int cbit[200], beg[200], dbit;
int maxparent[200],minparent[200],maxparent2[200],minparent2[200];
int quot,rem,modd,func,imaxt,imint,imax2,imin2,roop;
char string[10];
long mod[10];
int percent, kol;
int nbrep, nerep, nbmut, nemut, nbmutval, nemutval, nbmutmin, nemutmin,
    nbmutmv, nemutmv, nbcross, necross;

short wx, wy, xb, yb, lw, hw;


ldiv_t result;
div_t result2;
div_t result3;

void comparison(void);
void fitfun(double f(double *));
void refitfun(double f(double *));
void max1(void);
void min1(void);
void remax1(void);
void remin(void);
void intpat(void);
void reproduction(void);
void mutation(void);
void mutval(void);
void mutmin(void);
void mutminval(void);
void crossover(void);
void decigene(void);
void max_min (double* ord);
void crossover1(void);

int allocateGeneStorage(void);
void freeGeneStorage(void);

void setBitTo(int, int, int);
void setBit(int, int);
void clearBit(int, int);
void flipBit(int, int);
int bit(int, int);

void setChrof(int, int, double);
void clearChrof(int, int);
double chrof(int, int);

void setFitness(int, double);
double fitness(int);

int numberOfVariables(void);
double
   arguments[MAX_NUMBER_OF_ARGUMENTS],
   lowerBounds[MAX_NUMBER_OF_ARGUMENTS],
   upperBounds[MAX_NUMBER_OF_ARGUMENTS],
   tolerances[MAX_NUMBER_OF_ARGUMENTS];

// BEGINNING OF ADDITION
#pragma argsused
int CALLBACK LibMain (HANDLE hInstance,
   WORD wDataSeg,
   WORD wHeapSize,
   LPSTR lpszCmdLine)
{
   if (wHeapSize > 0)
     UnlockData(0);

   return 1;
}

#pragma argsused
int CALLBACK WEP (int nParameter)
{
   return 1;
}

int CALLBACK _export functionString( char far *functionString )  {
   return( initFKeyin( functionString ) );
}

// END OF ADDITION


double chrofToArgument(int index,double chrofValue)  {
   if( mod[index] == 0 )  {
      crashInt("minimize: Mod of %d is zero!",index);
   }
   return( lowerBounds[index] +
      (chrofValue / mod[index]) * (upperBounds[index]-lowerBounds[index])
   );
}

double f(double *);

void CALLBACK _export initializeMinimization(
   double passedLowerBounds[],
   double passedUpperBounds[],
   double passedTolerances[]
)  {
   int i;

   inform("Inside initializeMinimization");
   ngene = numberOfVariables();
   informInt("initializeMinimization: ngene: %d", ngene);
   for( i=0; i<ngene; i++ )  {
      lowerBounds[i] = passedLowerBounds[i];
   upperBounds[i] = passedUpperBounds[i];
   tolerances[i] = passedTolerances[i];
}
inform("initializeMinimization: copied arrays");

   /* This part determines the values of parameters such as
      the number of genes, the number of bits per gene and chromosomes. */
fac = 1.5;
cross = 5;

for( i=0; i<ngene; i++ )
   nbit[i] = log((upperBounds[i]-lowerBounds[i])/tolerances[i]) / log(2.0) + 1;

slength=0;
for( i=0; i<ngene; i++ )
   slength += nbit[i];

if( allocateGeneStorage() )  exit(1);

for( j=0; j<ngene; j++ )  {
   mod[j]=1;
   for( i=0; i<nbit[j]; i++ )
     mod[j]=mod[j]*2;
}
/* This part initializes all the matrix elements
   with logic value "0". */

     it2_rand=rand();
     result3=div(it2_rand,14);
     intn=result3.rem+10;
intn=100;

for( i=0; i<intn; i++ )
   for( j=0; j<slength; j++ )
      clearBit(i,j);

/* This part generates eight randomly selected
   initial genes. */

for( i=0; i<intn; i++ )
   for( j=0; j<ngene; j++ )
      clearChrof(i,j);

for( i=0; i<intn; i++ )
   for( j=0; j<ngene; j++ )  {
      int_rand=rand();
      result=ldiv((long)int_rand,mod[j]);
      setChrof(i,j,(double)result.rem);
   }
/* This part evaluates all matrix elements */

intm_pxval=0;

for( i=0; i<intn; i++ )  {
   intm_pxval=0;
   for( j=0; j<ngene; j++ )  {
      geneval = (int)chrof(i,j);
      for( t=0; t<nbit[j]; t++)  {
         k=intm_pxval + t;
         result=ldiv((long)geneval,(long)2);
         setBitTo(i,k,(int)result.rem);
         geneval=(int)result.quot;
         if( t==nbit[j]-1 )
          intm_pxval=intm_pxval+nbit[j];
      }
   }
}
fitfun(f);
```

```
   max1();
   min1();
   intpat();
   mold = maximum;
   minold = minimum;
   mold2 = maximum2;
   minold2 = minimum2;
}

int CALLBACK _export iterateMinimization( int properties[8] )  {
   // Answer 0 for normal, 1 for done, 2 or greater for erroneous return
   int doFuzzyLogic = 1;

   inform("Inside iterateMinimization");
   nb=0;
   ne=0;
   reproduction();
   comparison();
   if( doFuzzyLogic )  {
      if (dbit <= (int)(slength/fac))  {
   inform("iterateMinimization: doing crossover");
         crossover();
         crossover1();
         decigene();
         refitfun(f);
         remax1();
         remin();
         intpat();
      }  else  {
   inform("iterateMinimization: doing mutation");
         mutation();
         mutval();
         mutmin();
         mutminval();
         decigene();
         refitfun(f);
         remax1();
         remin();
         intpat();
      }
   }  else  {
      crossover();
      crossover1();
      mutation();
      mutval();
      mutmin();
      mutminval();
      decigene();
      refitfun(f);
      remax1();
      remin();
      intpat();
   }
   properties[0] = nb;
   properties[1] = ne;
   properties[2] = nbcross;
   properties[3] = necross;
   properties[4] = nbrep;
   properties[5] = nerep;
   properties[6] = nbmut;
   properties[7] = nemut;
   inform("iterateMinimization: set properties");

   if ((maximum == mold) && (minimum == minold) &&
       (imaxt == 0) && (imint == 2) && (imax2 == 1) && (imin2 == 3))  {
      return 1;
   }
   mold = maximum;
   minold = minimum;
   mold2 = maximum2;
   minold2 = minimum2;
   return 0;
}

void CALLBACK _export getResults(
   double locationOfMinimum[], double* valueAtMinimum,
   double locationOfMaximum[], double* valueAtMaximum
)  {
   int i;

   *valueAtMinimum = minimum;
   *valueAtMaximum = maximum;
   for( i=0; i<ngene; i++ )  {
      locationOfMinimum[i] = chrofToArgument(i,chrof(imint,i));
      locationOfMaximum[i] = chrofToArgument(i,chrof(imaxt,i));
   }
}

VOID CALLBACK _export terminateMinimization(void)  {
   freeGeneStorage();
}

void loadFitnessesTo(int loadSize, double f(double *))  {
   for( i=0; i<loadSize; i++ )  {
      for( j=0; j<ngene; j++ )
         arguments[j] = chrofToArgument(j,chrof(i,j));
      setFitness(i,f(arguments));
   }
}

void fitfun(double f(double *))  {
   loadFitnessesTo(intn,f);
}

void refitfun(double f(double *))  {
   loadFitnessesTo(ne,f);
}
void max1(void)
{
   maximum=fitness(1);
   imaxt=0;
   for( i=0; i<intn; i++ )
        {
         if (fitness(i)>=maximum)
        {
        maximum=fitness(i);
        imaxt=i;
        }
         else
        {
        maximum=maximum;
        }
        }
          if (imaxt==0)
        {
         maximum2=fitness(1);
         imax2=1;
        }
   else
        {
         maximum2=fitness(0);
         imax2=0;
        }
   for( i=0; i<intn; i++ )
        {
         if (fitness(i)>maximum2 && fitness(i)!=maximum && i != imaxt)
       {
        maximum2=fitness(i);
        imax2=i;
       }
         else
       {
        maximum2=maximum2;
       }
        }
}



void remax1(void)
{
   imaxt = 0;
   maximum = fitness(0);
   for( i=0; i<ne; i++){
         if (fitness(i) > maximum)
       {
        maximum = fitness(i);
        imaxt = i;
       }
         else if (fitness(i) == maximum)
       {
        imaxt = imaxt;
        maximum=fitness(i);
       }
         else
       {
        imaxt=imaxt;
        maximum=maximum;
       }
   }

   if (imaxt==0)
        {
         maximum2=fitness(1);
         imax2=1;
        }
   else
        {
         maximum2=fitness(0);
         imax2=0;
        }

   for( i=0; i<ne; i++ )
        {
         if (fitness(i)>maximum2 && fitness(i)!=maximum)
       {
        maximum2=fitness(i);
        imax2=i;
       }
         else
       {
        maximum2=maximum2;
       }
        }
}


void min1(void)
{
   imint = 2;
   minimum = maximum;
   for( i=0; i<intn; i++ )
       {
        if (fitness(i)<=minimum)
      {
       minimum=fitness(i);
       imint=i;
      }
        else
      {
       imint=imint;
       minimum=minimum;
      }
       }

   if (imint==0)
      {
       minimum2=fitness(1);
       imin2=1;
      }
   else
      {
       minimum2=fitness(0);
       imin2=0;
      }

   for( i=0; i<intn; i++ )
       {
        if (fitness(i)<minimum2 && fitness(i)!=minimum)
      {
       minimum2=fitness(i);
       imin2=i;
      }
        else
      {
       imin2=imin2;
       minimum2=minimum2;
      }
       }


}

void remin(void)
{
    imint=0;
   minimum = maximum;
   for( i=0; i<ne; i++){

        if (fitness(i)<minimum){
          minimum=fitness(i);
          imint=i;
       }
        else if(fitness(i)==minimum){
          minimum=fitness(i);
          imint=imint;
       }
        else{
          minimum=minimum;
          imint=imint;
       }

   }

   if (imint==0){
       minimum2=fitness(1);
       imin2=1;
   }
   else{
    minimum2=fitness(0);
    imin2=0;
  }


  for( i=0; i<ne; i++){
      if (fitness(i)<minimum2 && fitness(i)!=minimum){
       minimum2=fitness(i);
       imin2=i;
      }
      else{
       minimum2=minimum2;
       imin2 = imin2;
      }
  }

}

void intpat(void)
{
  for( j=0; j<slength; j++ )
      {
      maxparent[j] = bit(imaxt,j);
      minparent[j] = bit(imint,j);
      maxparent2[j] = bit(imax2,j);
      minparent2[j] = bit(imin2,j);
      }
}



void reproduction(void)
{
  /* This part of program is to perform the reproduction of the maximum
     fitness chromesome */
  nb=0;
  ne=4;
  neold=0;
  nbold=0;
  for( j=0; j<slength; j++ )
      {
       setBitTo(0,j,maxparent[j]);
       setBitTo(1,j,maxparent2[j]);
       setBitTo(2,j,minparent[j]);
       setBitTo(3,j,minparent2[j]);
       }
      nbrep = nb;
      nerep = ne;
      nbold=nb;
      neold=ne;

}


void mutation(void)
{
  /* This part of program is to perform mutation of the maximum
     fitness chromesome */
  nb=neold;                                     /* fred */
  ne=neold+slength;

  for( i=nb; i<ne; i++)
      {
       for( j=0; j<slength; j++ )
          setBitTo(i,j,
               (i-neold != j) ? maxparent[j] : !bit(0,j));
      }
  nbmut = nb;
```

```
   nbold=nb;
   neold=ne;
   nb=neold;
   ne=neold+slength;

   for( i=nb; i<ne; i++ )
      for( j=0; j<slength; j++ )
         setBitTo(i,j,
            (i-neold != j) ? maxparent2[j] : !bit(1,j));
   nbold=nb;
   neold=ne;
   nemut = ne;
}


void mutval(void)
{
   /* This part of program is to perform mutation of the maximum
      fitness chromesome */

   int denom;

   nb=neold;
   ne=neold+slength;

   for( i=nb; i<ne; i++ )
      for( j=0; j<slength; j++ )
         setBitTo(i,j,maxparent[j]);

   for( i=nb; i<ne; i++ )   {
      it2_rand=rand();
      result3=div(it2_rand,3);
      n=result3.rem+1;
      int_rand=rand();
      denom = slength-n-1;
      denom = (denom<1 ? 1 : denom);
      result=ldiv((long)int_rand,(long)denom);
      p=(int)result.rem+1;

      for( j=i-neold; j<=i-neold+n; j++ )    /* fred ? */
         flipBit(i,j);
   }
   nbmutval = nb;

   nbold=nb;
   neold=ne;                                  /* fred */
   nb=neold;
   ne=neold+slength;

   for( i=nb; i<ne; i++ )
      for( j=0; j<slength; j++)
         setBitTo(i,j,maxparent[j]);
   tt=0;
   for( k=0; k<ngene; k++ )   {
      for( i=nb+tt; i<=nb+tt+nbit[k]; i++ )
         for( j=tt; j<i-neold; j++)
            flipBit(i,j);
      tt=tt+nbit[k];
   }
   nbold=nb;
   neold=ne;
   nemutval = ne;
}


void mutmin(void)
{
   /* This part of program is to perform mutation of the maximum
      fitness chromesome */
   nb=neold;                              /* fred */
   ne=neold+slength;
   nbmutmin = nb;

   for( i=nb; i<ne; i++ )
      for( j=0; j<slength; j++ )
         setBitTo(i,j,
            (i-neold != j) ? minparent[j] : !bit(2,j));

   nbold=nb;
   neold=ne;
   nb=neold;
   ne=neold+slength;

   for( i=nb; i<ne; i++ )
      for( j=0; j<slength; j++ )
         setBitTo(i,j,
            (i-neold != j) ? minparent2[j] : !bit(3,j));

   nbold=nb;
   neold=ne;
   nemutmin = ne;
}

void mutminval(void)
{
   /* This part of program is to perform mutation of the maximum
      fitness chromesome */
   int denom;

   nb=neold;
   ne=neold+slength;
   nbmutmv = nb;

   for( i=nb; i<ne; i++ )
      for( j=0; j<slength; j++ )
         setBitTo(i,j,minparent[j]);

   for( i=nb; i<ne; i++ )  {
      it2_rand=rand();
      result3=div(it2_rand,3);
      n=result3.rem+1; n=2;
      int_rand=rand();
      denom = slength-n-1;
      denom = (denom<1 ? 1 : denom);
      result=ldiv((long)int_rand,(long)denom);
      p=(int)result.rem+1;
      for( j=i-neold; j<=i-neold+n; j++ )
         flipBit(i,j);
   }
   nbold=nb;
   neold=ne;
   nb=neold;
   ne=neold+slength;

   for( i=nb; i<ne; i++ )
      for( j=0; j<slength; j++ )
        setBitTo(i,j,minparent[j]);

   tt=0;
   for( k=0; k<ngene; k++ )  {
      for( i=nb+tt; i<=nb+tt+nbit[k]-1; i++ )
         for( j=tt; j<i-neold; j++ )
            flipBit(i,j);
      tt=tt+nbit[k];
   }
   nbold=nb;
   neold=ne;
   nemutmv = ne;
}


void crossover1(void)
{
   int  k, beg, p, q, pp;
   div_t result;
   int denom;

   for( pp = 1; pp<10; pp++ )  {
      result = div(rand(), cross);
      q = result.rem + 1;

      denom = slength-q-1;
      denom = (denom<1 ? 1 : denom);
      result = div(rand(), denom);
      p = result.rem;

      nb = neold;
      i = nb;

/*  CROSSOVER OF MAX and MAX2 */

      for( beg = 0; beg < slength - q + 1; beg++ )  {
         k = p;
         for( j = 0; j < slength; j++ )
            setBitTo(i,j,
               (j < beg || j >= beg + q) ? maxparent[j] : maxparent2[k++]);
         i++;
         k = beg;
         for( j = 0; j < slength; j++)
            setBitTo(i,j,
               (j < p || j >= p + q) ? maxparent2[j] : maxparent[k++]);
         i++;
      }
      ne = i;      /* fred? */

      neold = ne;
      nbold = nb;

/*  CROSSOVER OF MIN and MIN2 */

      for( beg = 0; beg < slength - q + 1; beg++)  {
         k = p;
         for( j = 0; j < slength; j++ )
            setBitTo(i,j,
               (j < beg || j >= beg + q) ? minparent[j] : minparent2[k++]);
         i++;
         k = beg;
         for( j = 0; j < slength; j++ )
            setBitTo(i,j,
               (j < p || j >= p + q) ? minparent2[j] : minparent[k++]);
         i++;

      }
      ne = i;
   }
   neold = ne;
   nbold = nb;
}
```

```
void crossover(void)
{
   /* This part of program is to perform the corssover between the maximum
      fitness chromesome and the minimum fitness chromesome */

   int denom;

   nb=neold;
   ne=neold+80;
   nbcross = nb;

   for( i=nb; i<ne; i=i+2 )  {
      itt_rand=rand();
      result2=div(itt_rand,cross);
      n=result2.rem+1;
      int_rand=rand();
      denom = slength-n-1;
      denom = (denom<1 ? 1 : denom);
      result=ldiv((long)int_rand,(long)denom);
      p=(int)result.rem;
      for( j = 0; j < kol; j++ )
         if (p == beg[i])
            if ((p + cbit[i]) > slength)
               p -= cbit[j];
            else
               p += cbit[j];

      for( j=0; j<slength; j++ )
         if (p <= j && j < p+n)  {
             setBitTo(i,j,maxparent[j]);
             setBitTo(i+1,j,maxparent2[j]);
         }  else  {
             setBitTo(i,j,maxparent2[j]);
             setBitTo(i+1,j,maxparent[j]);
         }
   }

   nbold=nb;
   neold=ne;
   nb=neold;
   ne=neold+100;

   for( i=nb; i<ne; i=i+2 )  {
      itt_rand=rand();
      result2=div(itt_rand,cross);
      n=result2.rem+1;
      int_rand=rand();
      denom = slength-n-1;
      denom = (denom<1 ? 1 : denom);
      result=ldiv((long)int_rand,(long)denom);
      p=(int)result.rem;
      for( j=0; j<slength; j++ )
         if (p <= j  && j < p+n)  {
             setBitTo(i,j,minparent[j]);
             setBitTo(i+1,j,minparent2[j]);
         }  else  {
             setBitTo(i,j,minparent2[j]);
             setBitTo(i+1,j,minparent[j]);
         }
   }
   nbold=nb;
   neold=ne;
   necross = ne;
}


void decigene(void)
{
   int kb, ke;
/* This part is to converet binary to decimal for each gene */

   for( i=0; i<ne; i++)
      for( j=0; j<ngene; j++ )
         clearChrof(i,j);

   for( i=0; i<ne; i++ )  {
      kb = 0;
      for( j=0; j<ngene; j++ )  {
         gendec=0.0;
         ke = nbit[j] + kb;
         for( k = kb; k < ke; k++ )  {
            gendec=(double)bit(i,k) * pow((double)2.0,
                 (double)(k-kb));
            setChrof(i,j,chrof(i,j) + gendec);
         }
         kb += nbit[j];
      }

   }
}

void comparison(void)
{
   int count, jbeg, flag = 0;

   i = 0;
   cbit[i] = 0;
   count = 0;
   dbit = 0;

   for( j = 0; j < slength; j++ )  {
      if (bit(imaxt,j) == bit(imax2,j))  {
         dbit++;
         count++;
         if (flag == 0)
            jbeg = j;
         flag = 1;
      }  else if (count >= 2)  {
         cbit[i] = count;
         count = 0;
         beg[i] = jbeg;
         jbeg = 0;
         i++;
         flag = 0;
      }  else  {
         jbeg = 0;
         count = 0;
         flag = 0;
      }
   }

   if ( count >= 2)  {
      cbit[i] = count;
      beg[i] = jbeg;
      i++;
   }
   kol = i;
}
```

APPENDIX 3
GA GUI FOR DYNAMIC LINK LIBRARY

```
/* Windows application program with Graphical User Interface
   that uses GA DLL routines.
   Revised and finalized by J. Kim, August 1997.
   Physical Optics Corporation, All Rights Reserved.   */

#define STRICT
#include <windows.h>
#include <windowsx.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "dlltest.h"
#include "minimize.h"
// #include "newdll.h"
#pragma warning (disable: 4068)

static char szAppName[] = "GA";
static HWND MainWindow;
static HINSTANCE hInstance;
int iSelect = 0;   // index of list box selection
char lpszBuffer[80], szNumGeneBuffer[80];

PSTR aList[] = {"-x1^2 + 2*x1 + 1", "x1^2 - 2*x1 + 1", "2*sin(x1) + cos(x1)"};

// The program starts here.  The Window is registered
// and created and the message translation is called.
#pragma argsused
int PASCAL WinMain(HINSTANCE hInst, HINSTANCE hPrevInstance,
                   LPSTR lpszCmdParam, int nCmdShow)
{
  MSG Msg;

  if (!hPrevInstance)
    if (!Register(hInst))
      return FALSE;

  MainWindow = Create(hInst, nCmdShow);
  if (!MainWindow)
    return FALSE;
  while (GetMessage(&Msg, NULL, 0, 0))
  {
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
  }

  return Msg.wParam;
}

// Registration of Window class
BOOL Register(HINSTANCE hInst)
{
  WNDCLASS WndClass;

  WndClass.style          = CS_HREDRAW | CS_VREDRAW;
  WndClass.lpfnWndProc    = WndProc;
  WndClass.cbClsExtra     = 0;
  WndClass.cbWndExtra     = 0; // DLGWINDOWEXTRA; // required for dlg window
  WndClass.hInstance      = hInst;
  WndClass.hIcon          = LoadIcon(NULL, IDI_APPLICATION);
  WndClass.hCursor        = LoadCursor(NULL, IDC_ARROW);
  WndClass.hbrBackground  = GetStockBrush(WHITE_BRUSH);
  WndClass.lpszMenuName   = "GA";  // Window won't be created w/o this.
  WndClass.lpszClassName  = szAppName;

  return RegisterClass (&WndClass);
}
```

```
/ Creation of Window
WND Create(HINSTANCE hInst, int nCmdShow)

  HWND hwnd;
  HDC hDC;

  hInstance = hInst;


  // HWND hwnd = CreateDialog(hInst, szAppName, 0, NULL);
  hwnd = CreateWindow(szAppName, "Genetic Algorithm DLL TEST",
                      WS_OVERLAPPEDWINDOW,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL, NULL, hInst, NULL);

  if (hwnd == NULL)
    return hwnd;


  // MIL: Write your one-time initialization code here
   ///////////////////////////////////////////////////////////////////////////

   // Allocate an application
   MappAlloc(M_DEFAULT,&MilApplication);
   // Disable MIL error message to be displayed as the usual way
   MappControl(M_ERROR,M_PRINT_DISABLE);
   // Retrieve previous hanler ptr and user handler ptr
   MappInquire(M_CURRENT_ERROR_HANDLER_PTR,&HandlerPtr);
   MappInquire(M_CURRENT_ERROR_HANDLER_USER_PTR,&HandlerUserPtr);
   // Hook MIL error on function DisplayError()
   MappHookFunction(M_ERROR_CURRENT,DisplayErrorExt,this);
   // Allocate a system
   MsysAlloc(M_SYSTEM_SETUP,M_DEF_SYSTEM_NUM,M_COMPLETE,&MilSystem);


  ShowWindow(hwnd, nCmdShow);
  UpdateWindow(hwnd);

  hDC = GetDC(hwnd);
  SetBkColor( hDC, RGB(0, 255, 255) );
  TextOut(hDC, 61, 210, "locationOfMinimum:     " , 23);
  ReleaseDC(hwnd, hDC);

  return hwnd;
}

// Message translation module

LRESULT CALLBACK _export WndProc(HWND hwnd, UINT Message,
                                 WPARAM wParam, LPARAM lParam)
{
  switch(Message)
  {
    HANDLE_MSG(hwnd, WM_CREATE, ga_OnCreate);
    HANDLE_MSG(hwnd, WM_DESTROY, ga_OnDestroy);
    HANDLE_MSG(hwnd, WM_COMMAND, ga_OnCommand);
    HANDLE_MSG(hwnd, WM_PAINT, ga_OnPaint);
    default:
      return ga_DefProc(hwnd, Message, wParam, lParam);
  }
}

#pragma argsused
BOOL ga_OnCreate(HWND hwnd, CREATESTRUCT FAR* lpCreateStruct)
{
  int i;
  static char *Titles[] = {"minimize", "maximize"};
static char *Params[] = {"-10.0", "10.0", "0.1"};
static char *ParamInfo[] = {"lowerBounds", "upperBounds", "tolerance"};
static char *ProgramInfo[] = {"Num Of Genes:", "Num Of GeneBit:",
                              "Chromosome Length:"};

hListBoxTitle = CreateWindow("static", "FunctionString LIST",
                   WS_CHILD | WS_VISIBLE | WS_BORDER | SS_CENTER,
                   50, 50, 200, 20, hwnd, NULL,
                   hInstance, NULL);
hParamInfoTitle = CreateWindow("static", "Parameters INFO",
                   WS_CHILD | WS_VISIBLE | WS_BORDER | SS_CENTER,
                   300, 50, 240, 20, hwnd, NULL,
                   hInstance, NULL);
hOutputTitle = CreateWindow("static", "Optimized Value",
                   WS_CHILD | WS_VISIBLE | WS_BORDER | SS_CENTER,
                   51, 180, 198, 20, hwnd, NULL,
                   hInstance, NULL);

for (i=0; i<3; i++)
  hParamInfo[i] = CreateWindow("static", ParamInfo[i],
                   WS_CHILD | WS_VISIBLE | WS_BORDER | SS_CENTER,
                   300, 70 + i*20, 140, 20, hwnd, NULL,
                   hInstance, NULL);
for (i=0; i<3; i++)
  hProgrammerInfoTitle[i] = CreateWindow("static", ProgramInfo[i],
                   WS_CHILD | WS_VISIBLE | WS_BORDER | SS_CENTER,
                   300, 220 + i*20, 140, 20, hwnd, NULL,
                   hInstance, NULL);
/*
for (i=0; i<3; i++)
  hProgrammerInfo[i] = CreateWindow("static", NULL,
                   WS_CHILD | WS_VISIBLE | WS_BORDER | SS_CENTER,
                   300 + 140, 220 + i*20, 100, 20, hwnd, NULL,
                   hInstance, NULL);
*/
hListBox = CreateWindow("listbox", NULL,
                   WS_CHILD | WS_VISIBLE | LBS_STANDARD,
                   51, 70, 198, 80, hwnd, ID_LISTBOX,
                   hInstance, NULL);
hOkButton = CreateWindow("button", "OPTIMIZE",
                   WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                   51, 280, 120, 40, hwnd, ID_OK,
                   hInstance, NULL);

for (i=0; i<3; i++)
  hEditBox[i] = CreateWindow("edit", Params[i],
                   WS_CHILD | WS_VISIBLE | WS_BORDER | ES_CENTER,
                   440, 70 + (i*20), 100, 20, hwnd, NULL,
                   hInstance, NULL);
hGroupBox = CreateWindow("button", "Optimization Mode",
                   WS_CHILD | WS_VISIBLE | BS_GROUPBOX,
                   300, 150, 240, 50, hwnd, NULL,
                   hInstance, NULL);
ButtonWindows[0] = CreateWindow("button", Titles[0],
                      WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON |
                      WS_TABSTOP,
                      320, 170, 90, 20,
                      hwnd, ID_MINIMAX,
                      hInstance, NULL);
ButtonWindows[1] = CreateWindow("button", Titles[1],
                      WS_CHILD | WS_VISIBLE | BS_AUTORADIOBUTTON
                      | WS_GROUP,
                      300 + 120, 170, 90, 20,
                      hwnd, ID_MINIMAX + 100,
                      hInstance, NULL);
for (i=0; i<3; i++)
  SendMessage(hListBox, LB_ADDSTRING, 0,
             (LPARAM)((LPCSTR)aList[i]));
  SendMessage(hListBox, LB_SETCURSEL, 0, 0L);

  Button_SetCheck( ButtonWindows[0], TRUE );


  return TRUE;
}

#pragma argsused
void ga_OnDestroy(HWND hwnd)
{
  PostQuitMessage(0);
}

#pragma argsused
void ga_OnCommand(HWND hwnd, int id, HWND hwndCtl,
                  UINT codeNotify)
{
   char functionStr[80], str1[80], str2[80], str3[80];
   double lowerBounds[1];
   double upperBounds[1];
   double tolerances[1];
   int result, done = 0, finalResult = 0;
   int props[8];
   double locationOfMinimum[1], locationOfMaximum[1];
   double valueAtMinimum, valueAtMaximum;
   double optimumLoc;
   // char   *minStr;
   int ngene, chromelength;
   HDC hDC;
   // int dec, sign, ndig = 5;


   switch(id)
   {
     case ID_OK:
       iSelect = (int)SendMessage(hListBox, LB_GETCURSEL, 0, 0L);
       SendMessage(hListBox, LB_GETTEXT, iSelect, functionStr);

       GetWindowText(hEditBox[0], str1, 10);
       GetWindowText(hEditBox[1], str2, 10);
       GetWindowText(hEditBox[2], str3, 10);

       lowerBounds[0] = atof( str1 );
       upperBounds[0] = atof( str2 ); // 10.0;
       tolerances[0] = atof ( str3 ); // 0.1;

       functionString(functionStr);

       ngene = numberOfVars();
       wsprintf(szNumGeneBuffer, "    %u", ngene);

       initializeMinimization(lowerBounds,
                              upperBounds,
                              tolerances);
       while (!done) {
         result = iterateMinimization( props );
         if ( result == 2 ) {
           finalResult = -1;
           done = 1;
         }
         else if ( result == 1 ) {
           done = 1;
         }
       }
       // if ( finalResult != 0 )
       //   return ( finalResult );
```

```
      getResults( locationOfMinimum, &valueAtMinimum,
                  locationOfMaximum, &valueAtMaximum );
      if ( OptMode == MINIMIZE ) {
         optimumLoc = locationOfMinimum[0];
         wsprintf(lpszBuffer, "locationOfMinimum: %d", (int)optimumLoc);
      }
      else if ( OptMode == MAXIMIZE ) {
         optimumLoc = locationOfMaximum[0];
         wsprintf(lpszBuffer, "locationOfMaximum: %d", (int)optimumLoc);
      }
      else {
         wsprintf(lpszBuffer, "Do Nothing!!!", NULL);
      }

      // SetWindowText(hEditBox1, lpszBuffer);

      terminateMinimization();

      InvalidateRect(hwnd, NULL, TRUE);
      break;

    case ID_MINIMAX:
      OptMode = MINIMIZE;

      hDC = GetDC(hwnd);
      SetBkColor( hDC, RGB(0, 255, 255) );
      TextOut(hDC, 61, 210, "locationOfMinimum:     ", 23);
      ReleaseDC(hwnd, hDC);
    break;

    case ID_MINIMAX + 100:
      OptMode = MAXIMIZE;

      hDC = GetDC(hwnd);
      SetBkColor( hDC, RGB(0, 255, 255) );
      TextOut(hDC, 61, 210, "locationOfMaximum:     ", 23);
      ReleaseDC(hwnd, hDC);
    break;

   }
}

void ga_OnPaint(HWND hwnd)
{
   HDC hDC;
   PAINTSTRUCT ps;
   RECT rect;
   int cxClient, cyClient;

   hDC = BeginPaint(hwnd, &ps);
   SetBkColor( hDC, RGB(0, 255, 255) );
   TextOut( hDC, 61, 210, lpszBuffer, strlen(lpszBuffer) );
   TextOut( hDC, 440, 220, szNumGeneBuffer, strlen(szNumGeneBuffer) );
   EndPaint(hwnd, &ps);

}
```